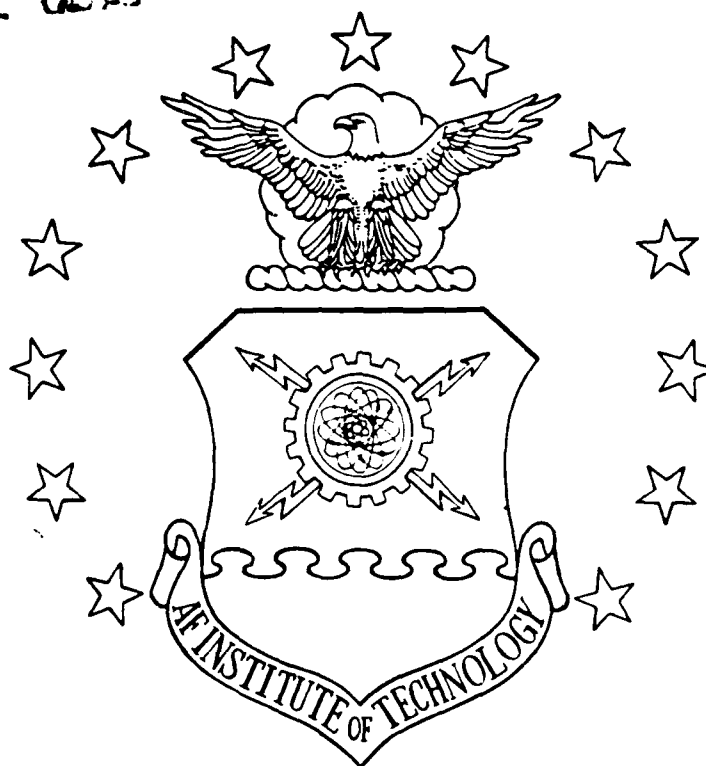AD-A202 794

AUTOMATED EVALUATION SYSTEM FOR THE JOINT

PLANNING (JPLAN) EXERCISE SYSTEM

THESIS

Chester J. Jean, Jr.
Captain, USAF

AFIT/GLM/LSM/88S-39

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

89   1   17 353

AUTOMATED EVALUATION SYSTEM FOR THE JOINT

PLANNING (JPLAN) EXERCISE SYSTEM

THESIS

Chester J. Jean, Jr.
Captain, USAF

AFIT/GLM/LSM/88S-39

DTIC
ELECTE
1 8 JAN 1989
D

The contents of the document are technically accurate, and no
sensitive items, detrimental ideas, or deleterious information is
contained therein. Furthermore, the views expressed in the
document are those of the author and do not necessarily reflect
the views of the School of Systems and Logistics, the Air
University, the United States Air Force, or the Department of
Defense.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	

COPY
INSPECTED
6

AFIT/GLM/LSM/88S-39

AUTOMATED EVALUATION SYSTEM FOR THE JOINT

PLANNING (JPLAN) EXERCISE SYSTEM

THESIS

Presented to the Faculty of the School of Systems and Logistics

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Logistics Management

Chester J. Jean, Jr., B.S.

Captain, USAF

September 1988

Preface

The purpose of this study was to develcp a computer program which would evaluate student performance in the JPLAN computer exercise used in the Air Force Institute of Technology (AFIT) Professional Continuing Education (PCE) Combat Logistics Course (LOG 299).

JPLAN is a simulation of the Joint Operations Planning System (JOPS), which is the Department of Defense system for conducting the joint planning procesa. The JPLAN exercise requires students to complete force planning for a fictitious, and partially completed Operations Plan.

This thesis discusses the background of the Joint Planning Process, the JOPS system, and JPLAN Exercise system. It then discusses the problems associated with the manual method of evaluating student performance, and finally explains how the evaluation computer program was developed.

This project was not a solo effort and several people have my gratitude for their help. I am deeply indebted to my thesis advisor, Capt Joel Melsha for her assistance in the functional areas of this project, and to Capt Mark Roth for his assistance in the technical areas of this project. Finally, my wife ████ deserves a great deal of thanks for her understanding during those many hours when my face was buried in the computer screen.

Chester J. Jean, Jr.

ii

Table of Contents

iv

List of Figures

v

AFIT/GLM/LSM/88S-39

Abstract

The purpose of this thesis was to develop a computer program which would apply the same evaluation criteria to student performance in the Joint Planning (JPLAN) exercise simulation system that the instructor previously had to apply manually.

JPLAN, which is used in Combat Logistics (LOG 299), an Air Force Institute of Technology (AFIT) Professional Continuing Education (PCE) course, is a computerized simulation of the real-world Joint Operation Planning System (JOPS). JOPS is the Department of Defense system used for conducting joint planning.

In JPLAN, students are presented with a partially completed Operations Plan based on a fictitious scenario in which planning for the defense of Iguana, an American ally, is underway. Students must complete the plan by programming Combat Support Forces and Combat Services Support Forces to support pre-programmed combat forces.

Following completion of the exercise, student performance is evaluated by the instructor for shortfalls and discrepancies. Factors evaluated are; sufficient housekeeping, supply, combat support group, and transient maintenance to support each base's population; sufficient STAMP, STRAPP, fuel, field maintenance, and munitions maintenance for all flying squadrons; sufficient

aerial port support to handle the average daily tonnage coming into each base; two hospitals in theater; proper sequencing of the UTCs that must arrive in a given sequence; and arrival timing for the UTCs that must arrive prior to the start of flying operations on Day D-3.

A computer program was developed to take the JPLAN data files that comprise the students output, and evaluate them, applying the criteria described above. With this program, evaluation was reduced from over 3 hours using the manual method, to approximately 30 minutes.

The program was written using Microsoft C as the programming language and INGRES as the database management system, the same developmental software used in the JPLAN exercise system.

AUTOMATED EVALUATION SYSTEM

FOR THE

JOINT PLANNING (JPLAN) EXERCISE SYSTEM

I. Introduction

General Issue

Combat Logistics (LOG 299), taught as an Air Force
Institute of Technology (AFIT) Professional Continuing
Education (PCE) course has, as one of its objectives, to
show how the wartime roles and responsibilities of logistics
managers are integrated into the larger context of U.S. Air
Force and Department of Defense (DOD) wartime preparations
(5:1). As a means to meeting this objective, the course
includes blocks of instruction on the Deliberate Planning
Process and the Joint Operation Planning System (JOPS)
(Terms and abbreviations are defined in Appendix A). In
order to apply what they learn in the Deliberate Planning
Process block, the students participate in a computerized
exercise called the Joint Planning (JPLAN) exercise.
JPLAN, originally developed by what is now called the Air
Force Wargaming Center, is also used at the Air Command and
Staff College (ACSC), and the Air War College (AWC). It is
a simulation of the real-world Joint Operation Planning
System Automated Data Processing (JOPS ADP) computer system
(9:2).

1

In the JPLAN simulation scenario, plan development for the defense of Iguana, a fictitious ally of the United States, is already underway with planning for four of the six bases in Iguana complete (2:3). The students are presented with a simulated intelligence report on the situation in Iguana, a limited amount of transportation resources, and a force list peculiar to JPLAN which contains the Unit Type Codes (UTC) of available forces to be assigned to specific bases in the defense of Iguana. The students are then tasked to complete the plan development. They must complete support force planning, complete the Time Phased Force Deployment Data (TPFDD), and plan for the movement of all forces in the force list to bases in Iguana.

Following initial plan development, the students must test the transportation feasibility of their plan. This involves making sure that airlift requested is available, and that all units are delivered to their Port of Debarkation (POD) no later than their Latest Arrival Date (LAD). The students then must resolve any shortfalls in supportability and transportation that they find. Some examples of shortfalls are: inadequate housekeeping to support the forces at a base, not enough aerial port capability to unload incoming cargo, not planning enough medical support to care for the forces in the theater of operations, or not moving Standard Air Munitions Packages (STAMP) and Standard Tank, Racks, Adapters, and Pylons

2

Packages (STRAPP) (Appendix A) with a tactical fighter squadron. They must also resolve any UTC sequencing and timing errors. There are a number of UTCs which must have a Latest Arrival Date (LAD) which will have them arrive at their Planned Operating Base (POB) in Iguana no later than Day D-3, and in a specified proper sequence.

In order to evaluate the students' performance in the exercise, the instructor must manually review the exercise data files for shortfalls such as those mentioned above. According to Capt Joel Melsha, the Combat Logistics Course Director, and Lt Col Dennis Dragich, Combat Logistics instructor and previous course director, this method is time consuming and not entirely accurate (7,11). The instructor must manually add and subtract many numbers from the data files, and keep track of these calculations, while looking for other numbers to sum and compare to those already summed. For example, the total force population at each base must be calculated. Then, the evaluator must locate housekeeping support, medical support, and base supply support selected by the team for each base. The number of forces each of these can support must be totaled, then those numbers must be compared to the actual base population. A number of UTCs must also be checked to ensure they are programmed in the proper sequence, and are programmed to arrive by D-3. The instructors say it is very easy to add the wrong numbers, lose a sum of numbers, or entirely miss some numbers. They believe that a more accurate method of

evaluation would be fairer to the students, and that a
faster method of evaluation would provide feedback to the
students in minutes, rather than hours, thus providing
stronger reinforcement for what they learned in the
exercise.

Specific Problem

The general issue described above is summarized in this
specific problem: How can a computer program be developed
to evaluate student output from the JPLAN exercise system
more accurately than the present manual method of
evaluation, and so reduce the time required to provide
feedback to the students on their exercise performance from
hours to minutes, lessening the instructor's workload?

Background

The Planning Process. The Air Force does not operate
alone in defending the United States, nor would it fight
alone. The Air Force is part of a team, along with the
Army, Navy and Marine Corps, and planning for war means that
the relationships and interactions between the team members
must be considered. Developing a plan in which more than
one service is involved, called joint planning, is a complex
process which requires a great deal of coordination between
the services. This is because of the large numbers and
different types of forces in the United States military, and
"the magnitude of the strategic transportation problem which

is involved in moving forces from their home bases to their Planned Operating Bases (POB)" (2:5.12).

The process which allows the services to develop joint plans is the Joint Operating Planning System (JOPS). JOPS is a DOD directed , Joint Chiefs of Staff (JCS) specified system designed to, "establish a set of procedures for global and regional operation plan development, review, and execution" (2:5.12). This means JOPS is a standard system for the services to use to develop and execute war plans. As the system designed for the joint planning process, JOPS is used by the specified and unified commands in developing their Operations Plans (OPLANs). JOPS "provides a standardized approach for planning and integrating these joint military operations" (9:2), which means that JOPS is the system that provides a standardized approach to developing plans which involve more than one service.

The entire planning process is a very complex set of interrelationships between several systems, involving a number of documents. To fully explain the planning process is beyond the scope of this research, so to facilitate understanding, a condensed explanation of the planning process and JOPS' part in it is presented here. The entire planning process can be viewed as a three link chain, as shown in Figure 1 (page 6), with JOPS on one end, the Planning, Programming and Budgeting System (PPBS) on the other, and the Joint Strategic Planning System (JSPS), in the middle (2:5.15).
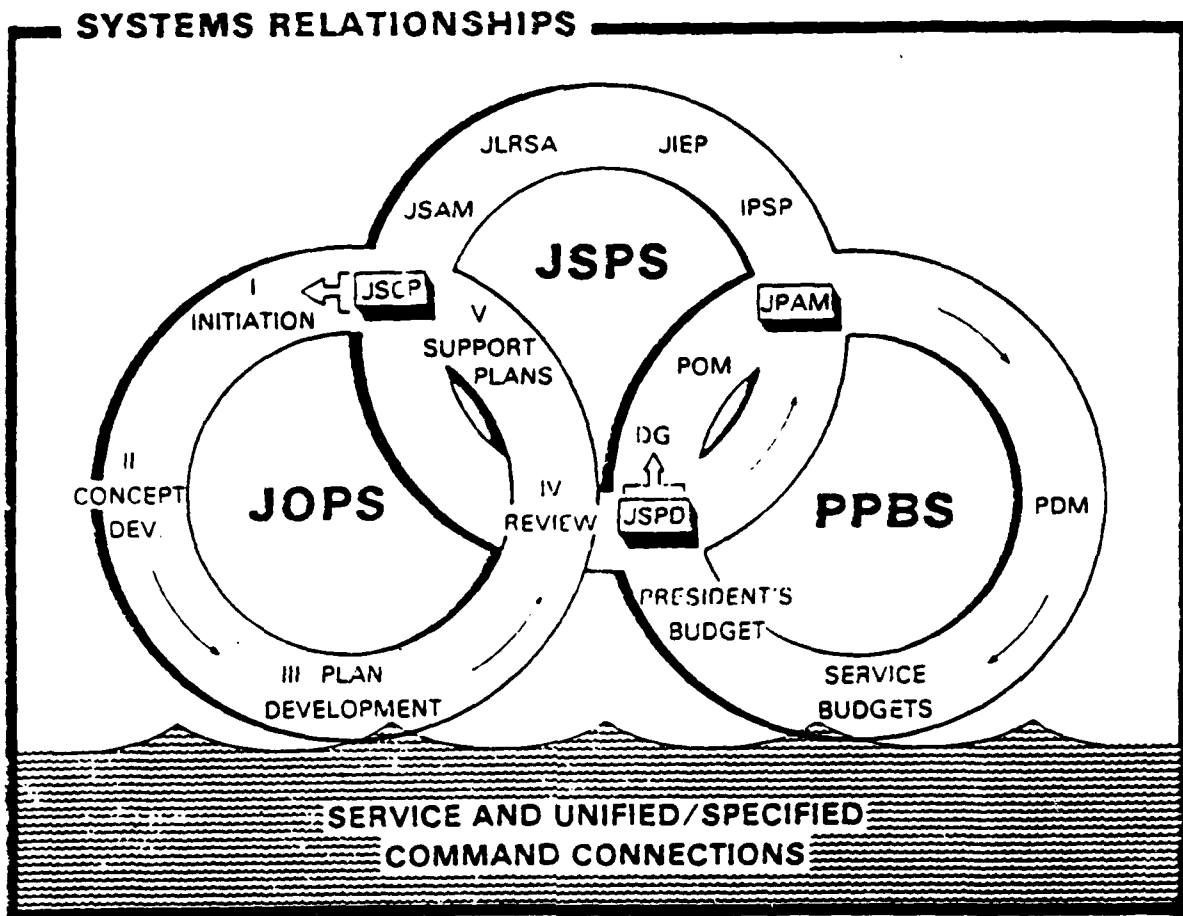
Figure 1. Planning Process Relationships (2:5-15)

The JSPS is a series of documents used by the Joint
Chiefs of Staff (JCS) to satisfy their strategic planning
responsibilities and identify force requirements based on
the perceived threat (2:5.4). The PPBS takes these force
requirements identified in JSPS, and generates the budget.
From its position in the middle of the chain, JSPS interacts
with both JOPS and PPBS. Documents from JSPS, which
identify force requirements, are input to PPBS, generating
budget requirements. Budgetary limitations may require
fewer forces than those requested through JSPS, so the

results of PPBS are fed back into JSPS. JSPS then makes changes in force requirements brought about by the budgetary constraints from PPBS. JSPS passes this updated force data, in the form of a Joint Strategic Capabilities Plan (JSCP) plan, to JOPS. In addition to availability of forces, the JSCP also contains intelligence information, and guidance issued by the Secretary of Defense. This information triggers the generation of Operations Plans (OPLANs) in JOPS to meet specific requirements identified by JSPS. Changes to the documents in JSPS affect both the PPBS process and the JOPS process, while changes in the PPBS process affect the JSPS documents, which in-turn, generates changes in JOPS. There is no direct JOPS and PPBS interaction (2:5.1-5.16).

JOPS ADP. JOPS is resident on the World Wide Military Command and Control System (WWMCCS) (2:5.12). WWMCCS is defined as "the system that provides the means for operational direction and technical administrative support involved in the function of command and control of U.S. military forces" (2:5.18). The computerized portion of JOPS, called JOPS Automatic Data Processing, or JOPS ADP, resides on the Honeywell 6000 (H6000) Series computer, which is the standard WWMCCS computer. These computers, which are usually found at Joint, Major Command (MAJCOM), and Numbered Air Force (NAF) headquarters in the Air Force, and equivalent levels of command in the other services, are joined together into the WWMCCS Intercomputer Network (WIN).

7

These connections allow organizations in the network to easily share databases, which occurs with regularity in JOPS ADP.

Deliberate planning is defined as "the cyclic process used when time permits the total participation of the commanders and staffs of the supported command..." (2:6.6). This refers to plans that are compiled during the normal planning cycle, and not in response to any contingency situation. The deliberate planning process, of which JOPS is a part, consists of five phases: initiation, concept development, plan development, plan review, and supporting plans (2:6.9). JOPS ADP is used in the plan development phase of the process.

The plan development phase contains eight steps which are described below and summarized in Figure 2 (page 9):

1. Force planning, in which all forces needed to support the concept of operations are identified.

2. Support planning, where support to sustain the forces in combat, such as supplies, medical material, civil engineering materials and replacement personnel, is planned.

3. Chemical/nuclear planning, where possible chemical and biological warfare is planned.

4. Transportation planning, where the planners begin the task of planning for the movement of the support.and combat forces previously identified.

5. Shortfall identification, where any shortfalls, particularly transportation shortfalls are identified and solved.

6. Transportation feasibility analysis, where transportation shortfalls are analyzed to determine if it is possible to move the forces selected to the correct base at the times selected.

7. Time-Phased Force and Deployment Data (TPFDD) refinement verifies that the units identified in the TPFDD are valid and that each OPLAN requirement has a unit assigned.

8. Documentation, in which the OPLAN is placed in its final format and published. (2:6.24-6.40).

PLAN DEVELOPMENT PHASE

STEP 1 -	FORCE PLANNING
STEP 2 -	SUPPORT PLANNING
STEP 3	CHEMICAL/NUCLEAR PLANNING
STEP 4 -	TRANSPORTATION PLANNING
STEP 5 -	SHORTFALL IDENTIFICATION
STEP 6 -	TRANSPORTATION FEASIBILITY ANALYSIS
STEP 7 -	TPFDD REFINEMENT
STEP 8 -	DOCUMENTATION
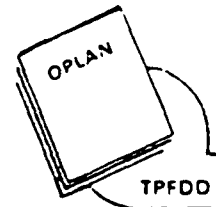
Figure 2. Steps in the Plan Development Phase (2:6-24)

In addition to deliberate planning, JOPS ADP also does time-sensitive planning, sometimes referred to as Crisis Action Procedures, which are conducted in crises situations. A crisis is defined as:

> An incident or situation involving a threat to the United States, its territories, and possessions that rapidly develops and creates a condition of such diplomatic, economic, political, or military importance to the U.S. Government that commitment of U.S. military forces and resources is contemplated to achieve U.S. national objectives [3:7-3].

In such situations, there is not sufficient time to follow the deliberate planning process. The portion of JOPS ADP that defines procedures to be used in time-sensitive planning is called the Crisis Action System (CAS). The Joint Deployment System (JDS) does the data processing for CAS (2:6.6).

Planners using JOPS ADP generate a TPFDD file which contains units identified for deployment by their UTC. The TPFDD file also contains characteristics of those units, such as: Port of Embarkation (POE), Port of Debarkation (POD), number of personnel, number of short tons (Appendix A) of cargo to be moved, movement dates and arrival dates (4:55). The TPFDD is then used as the basis to produce outputs from the JOPS ADP process. This output is in the form of OPLANS and Operation Orders. An OPLAN is a plan for "a single or series of connected operations to be carried out simultaneously or in succession" (2:5.12). This means that when a contingency arises with a scenario similar to the scenario an OPLAN, or several OPLANs were designed to

meet, those plans are executed. These plans are maintained in the JDS database, and saved for execution in the event of a contingency in which a specific OPLAN is applicable. An operation order is the "directive issued by a commander to subordinate commanders for the purpose of effecting the coordinated execution of an operation" (2:5.12), which means, an order to execute an OPLAN.

JPLAN. Combat Logistics (LOG 299), taught as an AFIT PCE course is designed to:

> provide logistics managers with an overview of many of the combat logistics plans, strategies, and procedures that will likely be implemented in a wartime scenario. It is designed to provide an understanding of how logistics contributes to the overall war effort and wartime requirements [5:1].

To meet these course objectives, Combat Logistics includes blocks of instruction in the Deliberate Planning Process, including JOPS, JSPS, and a Joint Planning (JPLAN) Exercise (5:i).

The Joint Planning (JPLAN) exercise is a simulation of JOPS ADP, involving the students in the plan development phase of the deliberate planning process (4:1). In JPLAN, students are presented with a fictional scenario in which an OPLAN for the defense of Iguana, an American ally, is in development, but not complete. In the OPLAN, combat forces have been selected for all six bases in theater, but combat support forces have been selected for only four of the six bases. The students must plan the remaining Combat Support (CS) Forces and Combat Services Support (CSS) Forces for the

OPLAN, and readjust combat support and services support where required.

JPLAN's database contains force list data similar in format to that found in JOPS. The students use this force list data to identify combat support and combat services support forces to deploy to Iguana. They then build a TPFDD file, similar to the one found in JOPS, using the forces they identified. Following this, just like JOPS, the TPFDD is used as the basis to identify support for the OPLAN.

The students must ensure they have programmed sufficient housing to shelter all incoming forces, construction units to erect any necessary housing, services units to feed the troops, sufficient medical personnel and medical supplies to meet the needs of the deployed force, aerial port units to on-load and off-load cargo, other forms of support as necessary, and that the support forces arrive in the proper sequence and on time (4:3).

Following force selection and loading into the computer database, the students then test the feasibility of movement of their selected forces to remote locations to meet OPLAN requirements. They must balance a limited number of transportation resources and base reception capability when moving their forces, yet they must still ensure the planned forces are delivered to their warfighting locations by the LAD. The students must then resolve shortfalls. See the General Issue section for examples of types of shortfalls.

JPLAN Rewrite. The original JPLAN exercise was
developed for ACSC by the Computer Science Division of the
Air University, now called the Air Force War Gaming Center.
As originally designed, JPLAN was written in FORTRAN and ran
on the H6000 computer, using a hardcopy thermal printer
terminal (9:2).

Combat Logistics' use of the original JPLAN exercise
required access to the AFLC H6000 computer through telephone
lines, a modem operating at 300 BAUD (most modems in
microcomputers operate at 1200 BAUD), and the Computational
Resources for Engineering and Simulation, Training, and
Education (CREATE) System (4:122). Using JPLAN in this
fashion caused problems. It was difficult to get sufficient
computer time to run JPLAN, response time was generally slow
because of many other users on the H6000, and data moved
slowly through the 300 BAUD modem.

Combat Logistics is regularly taught at other bases,
which caused additional problems in using the JPLAN game.
The host site did not always have access to an H6000
computer and the CREATE system, computer personnel to
install the software were not always available, the software
would not always run on the host computer, and the required
terminals were not always available (9:3). There was also
the time, expense and frustration of using long-distance
commercial telephone with a modem, which resulted in bad
connections and frequent disconnects.

In addition to problems with the host computer the original JPLAN exercise was not at all user friendly. All inputs and outputs were accomplished through the hardcopy terminal, which is very slow when compared to a standard video terminal. Because of the use of FORTRAN to write the program, commands were entered in a string, with commas separating each command. If an error occurred during input, the entire line had to be entered again. The students were spending time that should have been spent on the learning objective trying to make JPLAN run (9:3).

The original JPLAN system was re-written by Capt James Jansen, an AFIT School of Engineering 87-D graduate. Capt Jansen had several objectives which would make his JPLAN rewrite an improvement over the original JPLAN. One improvement was to make a system that could run on a microcomputer. With the current availability of microcomputers in most military organizations, compatibility and availability problems would be virtually eliminated. Another of Capt Jansen's objectives was to make the system more user friendly. He accomplished this by using on-screen menus in which system commands are entered. Capt Jansen also included help screens to answer users questions about JPLAN commands without the user having to refer to a manual. Finally, he wanted to utilize a state-of-the-art Database Management System (DBMS) in JPLAN, to make the new exercise more like JOPS ADP than the old JPLAN, and to make database updates easier for the instructors (9:5).

Capt Jansen chose PC INGRES by Relational Technology as the DBMS because the Air Force Wargaming Center, who would maintain JPLAN after its development and implementation, uses the mainframe version of INGRES in other projects they are developing. This meant lower training requirements for Wargaming Center personnel, and ensures compatibility with any other systems JPLAN may have to interface with in the future (9:35).

JPLAN was rewritten in the C computer language. C is a middle-level, block structured language, and is very portable. Portable means that a program written in C on one type of computer can easily be made to run on another type of computer. The C computer language was chosen for its compatibility with PC INGRES. The only other computer language compatible with PC INGRES is COBOL, but microcomputer versions of COBOL are relatively expensive. Additionally, COBOL is not as popular a computer language as it once was because programmers now consider it wordy and cumbersome (14:6).

The new JPLAN is now being used successfully at AFIT. According to the Combat Logistics Course Director, more students are successfully completing the JPLAN exercise objectives in the course than when the old JPLAN was used. She attributes this to the fact that the new system is more user friendly than the previous system because of the on-screen help, on-screen menus, and on-screen command boxes.

15

The students no longer have to spend their class time learning how to make JPLAN run. In addition, since no modem is needed, and there are no competing users, response times are significantly faster with the new system, further reducing wasted class time.

A way to improve the new JPLAN exercise would be to develop a system that would automatically evaluate the students' performance. In order to perform that evaluation, the instructor must print several reports from the JPLAN data base, and visually scan them, looking for particular conditions based on choices the students made in assembling their forces. The instructor must also perform many manual mathematical calculations. This process is time consuming, and it is possible for the instructor to make mistakes.

The judgements the instructor makes in evaluating the output could be converted into computer programming logic, and performed automatically. This process would be more accurate and much faster than the present method. The students would be able to receive feedback on their performance in minutes, compared to waiting overnight as with the present method. This would enhance the learning process by providing immediate reinforcement to their actions in the exercise.

Therefore, a computerized means of evaluating JPLAN exercise performance would benefit both the instructor and the students. In addition, since JPLAN is used at other

service schools, a computerized evaluation system would benefit them as well.

Research Objective

The objective of this research project was to improve the evaluation process of students' performance on the JPLAN exercise system by developing a computer program that would apply the same evaluation criteria used by the instructor. This program will perform the evaluation faster and more accurately than present manual method.

Investigative Questions

The following investigative questions were addressed in designing, developing, testing, and implementing the JPLAN exercise evaluation system:

1. What are the characteristics of the JPLAN exercise system?

2. What characteristics must the evaluation system have?

3. How can the instructor's evaluation criteria be converted to computer logic?

4. How should the output be formatted, and how will the output form, screen or hardcopy, affect the desired output format?

5. There is a senior service school, and a possible real-world application for JPLAN and the evaluation system. What features should be designed into the evaluation system to make it as compatible as possible with JPLAN, and to make the conversion to these other applications easier?

17

6.   Can the system be developed to be better than the existing manual evaluation system? If so, how?

Scope and Limitations

Following were the scope and limitations for this research project:

1.   The instructor for any course in any school using JPLAN must have the ability to modify the input data in the database to meet their course needs. This meant it was possible that there was a different database for every site using JPLAN. This system was developed with the database used at AFIT. Any tailoring of the evaluation system database to make it run at other sites will have to be done by that site.

2.   To ensure complete compatibility with JPLAN, the evaluation system was written in the same computer language as JPLAN, and used the same database structure that JPLAN uses.

3.   The majority of the research effort on this project was expended learning the computer language the evaluation system was written in, and the data base management system it used.

Assumptions

The following assumptions were applicable to this research project:

1.   Evaluation criteria were standard at all sites using the JPLAN exercise.

2.   The person using the evaluation system was familiar with the JPLAN system.

3.   Students would not use the evaluation system.

4.   Testing the evaluation system with the AFIT database would be sufficient to ensure compatibility with JPLAN at other sites.

5.   Following successful completion, the evaluation system would be incorporated into JPLAN, which is owned by the Air Force Wargaming Center.

6.   There would be no software changes to JPLAN which would affect the evaluation system while software for the evaluation system was being developed.

7.   The evaluation criteria would be able to be converted to computer program logic.

8.   Technical support on the development language, and the Database Management System (DBMS) would be available during software development.

II. Methodology

Overview

This chapter identifies the processes used to answer the investigative questions identified in Chapter I. Each investigative question will be repeated, and the specific methodology used to solve it will be presented.

Specific Methodology

The following methods were used to answer each investigative question.

Investigative Question 1.

> What are the characteristics of the JPLAN exercise system?

This question was partially answered in unstructured interviews with Capt Jansen (10), the author of the re-written JPLAN system, and Lt Col Dragich (7), the former Combat Logistics course director. A literature review of Chapter 2 of the JPLAN Documentation Manual (8:3-15); Chapter III, and Appendix B of Capt Jansen's Thesis (9:22-34,64-67); and the JPLAN computer source code, provided information on the format of the input files.

Learning the logic of the JPLAN system was accomplished through simulation and experimentation. This was done by running the JPLAN system to see what it does; stepping through the source code line-by-line; and re-running JPLAN frequently while stepping through the source code to ensure

20

the logic was understood. Both the source code, and the executable object code were available to accomplish this. Additional help in understanding the program logic was provided by Capt Jansen's thesis advisor in the AFIT School of Engineering, Capt Mark Roth.

Learning about the output from JPLAN, both hardcopy and data files, was achieved through a literature review, and experimentation with the system. Chapter 7 of the JPLAN Users Manual, Appendix A of Capt Jansen's thesis, and Chapter 2.4 of the JPLAN Documentation Manual, discuss program output, including how to generate output products, and how they are formatted. Experimenting with the system involved running the exercise, trying all system options, and taking all the output options to generate reports so they could be reviewed.

Investigative Question 2.

> What characteristics must the evaluation system have?

This question was answered in unstructured interviews with Capt Jansen (10), Lt Col Dragich (7), and Capt Roth (13). These interviews attempted to find the characteristics for the evaluation system that would best suit the course director's needs.

Investigative Question 3.

> How can the instructor's evaluation criteria be converted to computer logic?

This question was answered by conducting unstructured interviews with the instructors who regularly evaluate the

students' JPLAN performance, Lt Col Dragich (7) and Capt Melsha (11). The interviews attempted to determine if the evaluation was based on subjective or objective criteria.

If the initial questioning indicated that the instructors apply objective criteria, that meant they have specific rules they use when they evaluate JPLAN output. The instructors were asked to verbalize, or write down all the rules they apply when they are evaluating JPLAN output.

If the instructors indicated that they apply subjective evaluation criteria, that meant they don't have specific rules they follow when they evaluate JPLAN output, but base their evaluation on their opinions about the output, or some internal criteria. They were then asked to quantify, as best as they could, what they look for. Then that subjective evaluation criteria could be converted to firm rules that could be used to formulate either/or logic needed for a computer program.

Investigative Question 4.

> How should the output be formatted, and how will the output form, screen or hardcopy, affect the desired output format?

The Course Director was interviewed (11) to determine how the output should be formatted, and whether it should be screen or paper output. If paper output was desired, it had to be determined what width paper the printer to be used would have. This was necessary because the width or paper used would have an effect on the format of the output. For

22

example, a printer with a 10" carriage allows 80 characters of print on a line, while a printer with a 15" carriage allows up to 132 characters on a line.

Investigative Question 5.

> There is a senior service school, and a possible real-world application for JPLAN and the evaluation system. What features should be designed into the evaluation system to make it as compatible as possible with JPLAN, and to make the conversion to these other uses easier?

Unstructured interviews with Capt Roth (13) provided initial suggestions on how to ensure the evaluation system was fully compatible with JPLAN.

To make it fully compatible, the evaluation system was written in the same computer language as JPLAN, utilizes the same DBMS as JPLAN, and runs on the same computers as JPLAN, to make it fully compatible. Comments were placed in each routine of the program code to explain what that particular routine does. This will make the job of program maintenance easier for the person responsible for that task, by allowing that person to read the comments about what a program routine does, rather than reading the code and trying to decipher the program logic.

Investigative Question 6.

> Can the system be developed to be better than the existing manual system? If so, how?

Answering this investigative question required modeling and experimentation, but before these processes could begin, all previous investigative questions had to be answered. At this point, program design began. This required compilation

23

of all knowledge acquired thus far in the research project, such as: JPLAN system characteristics, the computer language the evaluation system will be written in, the DBMS the evaluation system will use, the instructor's needs, and the instructor's evaluation criteria. All of this information was used to determine the system design. Functional questions that came up during this period were answered by the course directors and technical questions were answered by Capt Roth.

The system design was then used as the basis to develop the computer language source code. Following coding, the program was compiled from source code to executable object code. This compilation process identified any syntax errors in the code. After all syntax errors were corrected, and the program had been successfully compiled, the program was tested in order to identify any logic errors. Four sets of output files from Combat Logistics student groups, along with the instructor's evaluation of each, were saved. The evaluation program was run using all of the test data files, and the output was compared to the instructor's evaluation. After all logic errors detected using this method were corrected, the program was tested again using live output data, and compared to the instructor's evaluation results. This live data testing process was repeated until the course director and the programmer were satisfied the program was error-free. Implementation simply involved supplying the

Combat Logistics course director and the Air Force War
Gaming Center with the program, and all associated
documentation. Since students will not use the evaluation
system, it will run separately, and will not need to be
included in the JPLAN installation.

III. Program Design, Development and Implementation

Overview

This chapter describes how the JPLAN evaluation system was developed. It will discuss the criteria that determined what the system design would be, what the system design actually was, how the program was developed and tested, and how the program was implemented.

Program Design

Programming Language and DBMS. The inherent JPLAN exercise program language and DBMS were the primary factors in determining the language and DBMS that the JPLAN evaluation system would use. Because JPLAN is written in C, using embedded Structured Query Language (SQL) from PC INGRES as the DBMS, it was determined that the evaluation system should be written in C and PC INGRES also to ensure complete compatibility with JPLAN. Compatibility is necessary to ensure that the evaluation system can use the JPLAN database, and to ease program maintenance. The Air Force War Gaming Center will be responsible for program maintenance of the evaluation system, as they are for JPLAN. Having the evaluation system written in the same language and DBMS as JPLAN makes that job easier, since the programmers at the War Gaming Center will only need to know one language.

26

Before any further program design could proceed, time had to be spent learning C and INGRES. This was accomplished by studying C and INGRES reference manuals, and writing several small programs.

Evaluation Criteria. Following the selection of the programming language and the DBMS, the Evaluation Criteria, which are the things the Combat Logistics instructor looks for when evaluating JPLAN output, became the driver for the overall program design.

There are a many criteria which the instructor considers when evaluating JPLAN output that had to be included in the evaluation system. First, the program had to ensure the final Army population in the entire theater is equal to the preloaded Army population, and that a minimum of two Modular Air Transportable Hospitals (MATH) are programmed for the theater. It had to make a number of checks in which supportability is based on base population. For example, it had to ensure there was enough housekeeping, supply support, combat support group (if a Harvest Eagle kit is programmed), and transient maintenance to support the population at each base. The program had to ensure there is not a Harvest Eagle and a Harvest Bare housekeeping set programmed for the same base. If Harvest Bare housekeeping support is planned for a base, the program had to make sure there is sufficient Maintenance/Operations Support Shelters (MOSS) support for each Tactical Fighter Squadron planned for the base, that a Harvest Bare Advance UTC is programmed,

27

and that it is programmed to arrive before the Harvest Bare
kit. The system had to ensure that there is sufficient
Standard Tanks, Racks, Adapters, and Pylons Packages
(STRAPP) and Standard Air Munitions Packages (STAMP) for
each Tactical Fighter Squadron that requires them and ensure
that there is sufficient fuel support for every flying
squadron planned for a base. It also had to ensure that the
aerial port capability programmed for each base is capable
of handling the average daily tonnage scheduled for off-
loading.

In addition to the supportability checks mentioned
above, the program also had to check UTC sequencing and
timing. There are a number of UTCs that must arrive in a
given sequence, and before a given D-Day. In addition,
there are a number of other UTCs that must be checked for
arrival by a given Day.

Close examination of the evaluation criteria revealed
that there were two general types. One required that two
numbers be compared to each other to see if one is less than
or equal to another. For example, the final Army population
is compared to the preloaded Army population to see if it is
less, which would indicate to the instructor that an Army
UTC was deleted from the database by the students. Another
is that some UTCs must occur in pairs or groups, or, the
presence of one UTC requires the presence of one or more
other UTCs. An example of this is if there is a Harvest

Eagle kit programmed for a base, there must also be a Combat Support Group planned for that base. This discovery made the rest of system design and development easier because it allowed all program sub-routines that performed criteria evaluations to be very similar to one of two types of evaluation criteria.

The specific evaluation criteria are as follows:

Army Preload. When the students complete the JPLAN exercise, they must have the same population of Army troops in the database as was in the database when the exercise started.

MATH. Two MATHs must be programmed within the theater. They can be programmed for any base(s) in the theater.

Housekeeping. There must be sufficient housekeeping support planned for each base to support no less than 80% of the programmed base population. Support planned for more than 80%, or even more than 100% is acceptable, there is no upper limit. This support can be either Harvest Eagle kits or Harvest Bare kits, but only one type is permitted at each base. A base cannot have both a Harvest Bare kit and a Harvest Eagle kit planned.

Combat Support Group. If Harvest Eagle support is planned for a base, there must be combat support group support planned to support no less than 90% of the base population. There is no upper limit.

Harvest Bare Advance. If Harvest Bare support is planned for a base, a Harvest Bare Advance UTC must be planned to arrive at the base before the Harvest Bare kit arrives. The Advance kit's LAD must be at least one day earlier than the Harvest Bare's LAD.

MOSS. If Harvest Bare support is planned for a base, MOSS support must also be planned for each Tactical Fighter unit also planned for that base. There must be the proper number of MOSS units, and they must be of the proper type for the type of aircraft planned for the base.

Aerial Port Capacity. The aerial port capacity planned for a base must be able to handle at least 75% of the average daily tonnage coming into the base prior to D-Day. The average daily tonnage coming into a base is computed by dividing the total amount of tonnage planned for the base by the number of days there is material actually arriving at the base. For example, if out of a 7 day time period, cargo is only arriving at a base during 5 days, the total tonnage would be divided by 5 not 7.

Supply. Supply support must be planned to support no less than 90% of the planned base population.

Fuel. There must be enough fuel support at a base to support every flying squadron planned for that base.

Transient Maintenance. There must be enough transient maintenance support planned for each base to support no less than 90% of the base population.

STAMP and STRAPP. For each Tactical Fighter Squadron that requires these, the proper type of STAMP and STRAPP for the aircraft type must be deployed.

UTC Sequencing and Timing. There are a number of UTCs that must arrive before Day D-3 to ensure they are in-place before flying operations begin. In addition to this, there are a number of units that must arrive in a prescribed sequence.

Database Design. After analyzing the evaluation criteria, it became apparent that there would have to be a way to allow the evaluation system to distinguish between types of UTCs and their properties so it could make the various evaluations. For example, the system would have to be able to tell the supply support UTCs from other UTCs, and what populations each supply support UTC could support; or for a tactical fighter squadron UTC, what STAMP and STRAPP UTCs are needed.

It was determined there were two ways to pass this type of information to the program. One was to include the UTCs and their properties in the program code. The other way was to design a data file for each type of UTC that needed to be evaluated in the program and include them as part of the JPLAN database.

Including the data in the program code would have made program development easier, and would have made the program run faster due to fewer database accesses that would have been required. However, for ease of maintainability, it was

decided to design and create data files for each type UTC
that needed to be checked by the program.

If the evaluation criteria data was included in the
program code, it would take a programmer to make any
necessary data changes. Including the information on
individual data files ensures that the user can update the
evaluation criteria data files through the INGRES
Interactive Query-By-Forms sub-system, rather than having to
have a programmer make the changes. Using INGRES in the
interactive mode would also allow the user to view the data,
which would not be possible if the data was in the program
code. The decision to include this information in data
files rather than in the program also allows each site that
will use the evaluation system to use slightly different
data if they desire.

Program Layout. A general program layout, based on
everything that was known about the programming language,
DBMS, the evaluation criteria, and the database was selected
next. The evaluation system would consist of one program
with a main sub-routine and a series of smaller sub-
routines, rather than several smaller programs linked
together. This was because the program would be small
enough to be easily handled as a single program, making
development and follow-on maintenance easier.

It was determined that the main sub-routine would
include the program introduction, and all user interface,

32

such as what output form is desired. It would also include all the calls to the other sub-routines. These other sub-routines would do such things as build tables, add numbers together, and make the actual evaluations. Each sub-routine will be discussed in detail in the Program Development section of this chapter.

Program Development

Database Construction. After program design was accomplished, construction of the new data files for the evaluation system was the next requirement. A total of 20 new files were added to the JPLAN database. The interactive JPLAN system was used to create and add records to these files. The interactive JPLAN system provides a complete means of database maintenance. It allows file creation and deletion, updating of existing records in a database, and inserting or appending new records to a database. A brief description of each new data file follows. Appendix C contains detailed record layouts of each data file, and information on file maintenance.

A PORT. The A_PORT file contains every aerial port UTC and its daily tonnage capability. It is also used to determine Aerial Port arrival timing and sequence.

STMP. This file contains every STAMP UTC, and the UTC of every tactical fighter squadron that uses that STAMP. Each record contains one STAMP UTC, and up to four aircraft squadron UTCs that use that particular STAMP.

STRP. This file contains every STRAPP UTC, and the UTC of every tactical fighter squadron that uses that STRAPP. Each record contains one STRAPP UTC, and up to three aircraft squadron UTCs that use that particular STRAPP.

EAGLE. This file contains the Harvest Eagle UTC, and the population it is capable of supporting. At the present time, there is only one Harvest Eagle UTC, but by putting this information on a file, it allows for new records that may be added in the future, such as the creation of UTCs for 550 or 275 man Harvest Eagle support kits.

BARE. This file contains every Harvest Bare UTC, and the population each is capable of supporting.

SUPPLY. This file contains every supply support UTC, and the population each is capable of supporting. It is also used in determining Supply arrival timing.

FUELS. This file contains every fuel/POL support UTC, and the number of flying squadrons each is capable of supporting. It is also used in determining Fuel support arrival timing.

CSG. This file contains every Combat Support Group UTC, and the population each is capable of supporting. It is also used in determining Combat Support Group arrival timing.

T MAINT. This file contains every transient maintenance UTC, and the population each is capable of

34

supporting. It is also used in determining Transient

Maintenance arrival timing.

MATH. This file contains all Modular Air

Transportable Hospital (MATH) UTCs.

ARMY. This file contains one record, which holds

the total of the preload Army population on the JPLAN

database.

ADV. This file contains the Harvest Bare advance

UTC.

HB MOSS. This file contains aircraft squadron

UTCs that need MOSS support, and the UTCs of the MOSS units

they need for support. Each record contains up to three

aircraft UTCs, up to four UTCs for MOSS units that will

support those aircraft, and the number of squadrons each

MOSS unit will support.

FLY SQN. This file contains the UTC of every

flying squadron. It is also used in determining flying

squadron arrival timing.

SURVEY. This file contains the UTCs of Site

Survey teams. It is used in checking timing and sequencing

of Site Survey teams.

BEEF. This file contains the UTCs of Prime Beef

and Red Horse units. It is used in checking timing and

sequencing of Prime Beef and Red Horse units.

SECURITY. This file contains the UTCs of Security Police units. It is used in checking timing and sequencing of Security Police units.

COMM. This file contains the UTCs of Communication units. It is used in checking timing of arrival of Communications units.

FMAINT. This file contains the UTCs of aircraft units requiring Field Maintenance support, the aircraft type, and the UTCs of the Field Maintenance units. It is used in determining arrival timing of Field Maintenance units.

MMAINT. This file is similar to the FMAINT file, except it contains the UTCs of Munitions Maintenance units, and the UTCs of aircraft units that need them. It is used in determining arrival timing of Munitions Maintenance units.

In addition to the new data files mentioned above, the evaluation system would use several of the existing JPLAN data files for input. One is the AIR_TPFDD file. This file contains all UTCs that were programmed for deployment during the exercise, their Planned Operating Base (POB) and their LAD. Cross referencing of UTCs in this file with UTCs contained in the files discussed above is necessary in the program to make the required checks. It is used to ensure required UTCs are present, and to compute the number of times they occur per base to assist in making the evaluation.

Another JPLAN file used for input is the AIRBASE file. The evaluation system performs its evaluation by base. It reads a record from the AIRBASE file, evaluates that base, then processes the next base. This process continues until the end of the AIRBASE file is reached.

The M10POD JPLAN file contains personnel and tonnage information in LAD and airbase code sequence. This file is used for computing the average amount of tonnage coming into a base, and in adding up the each bases total population, the theater-wide Army population.

Program Code. After construction of the new data files, actual program coding began. The first section of the program, The Declare Section, identifies working and holding areas to the program (See Appendix B – JPLAN Evaluation System Program Code). These working areas include JPLAN database files that the program will use, array structures which will hold data read from the JPLAN files, and other working areas which will hold numbers and characters during processing.

Coding the main sub-routine was next. It does not do any actual processing, but calls each sub-routine that does actual processing. It first asks the user whether they desire Hardcopy or Screen output, or Both.

After this step, the main sub-routine calls a series of sub-routines that build arrays from many of the data files used by this program. This was done to reduce the number of

times the system must read the database while it makes its evaluations. Arrays are stored in the computer's memory while the program is running, while the database is stored on disk. The computer can access its memory faster than it can access a disk, so for those files where multiple accesses would have been necessary, arrays are constructed to speed processing.

Within the main sub-routine, there is a loop that is executed once for every base in the AIRBASE file. Except for the Army preload and the MATH checks, which are done once for the whole theater, every evaluation criteria is accomplished once for every base on the AIRBASE file. Within the loop, each base population and the average daily amount of tonnage coming into that base are totaled, and a sub-routine which performs each of the evaluation criteria discussed in the Evaluation Criteria section is performed. When the loop has been executed once for every base in the AIRBASE file, a message saying the evaluation is complete is displayed, and the program ends.

The evaluation criteria discussed earlier in this chapter are performed in separate sub-routines. The programming logic behind each of the criteria is as follows:

Army Preload. The objective here is to make sure the Army population at the end of the exercise is not smaller than preload Army population. The preload population total is first retrieved from the Army file. Next the ending Army population is computed by adding all of

38

the 'ARMY' fields from the M10POD file. That number is compared to the preload Army population, and if it is smaller, an error message is displayed.

MATH. MATH UTCs are retrieved from the MATH array, and a total of all occurences of all MATH UTCs in the AIR_TPFDD file is computed. If, after all MATH UTCs have been processed, the total of MATHs programmed is less than two, an error message is displayed.

Supply. Each supply UTC from the SUPPLY array checked against the AIR_TPFDD file for number of occurences. That number is multiplied by the population that supply UTC can support, and then added to a running total. After all supply UTCs have been processed, the running total is multiplied by 1.1 and compared to the base population. If it is smaller, an error message is displayed.

Transient Maintenance. This sub-routine works like the supply support check, using UTCs from the T_MAINT array.

Aerial Port Capacity. Each aerial port UTC is read from the A_PORT array and checked against the AIR_TPFDD file for the number of times the UTC occurs in the AIR_TPFDD file. The number of occurences is multiplied by the amount of cargo that the UTC can handle, and added to a running total. Following the check of all aerial port UTCs, the running total is multiplied by 1.25, and is compared to the

average daily tonnage the base can handle. If the running
total is less, an error message is displayed.

Housekeeping Support. First, the AIR_TPFDD is
checked for the presence of a Harvest Eagle. The Harvest
Eagle UTC is retrieved from the EAGLE array, and the number
of times it occurs in the AIR_TPFDD file is added. That
number is then multiplied by the population that a Harvest
Eagle can support.

A check for a Harvest Bare kit is performed next.
Each Harvest Bare UTC is retrieved from the BARE array. The
number of times each UTC occurs in the AIR_TPFDD file is
added, multiplied by the population that UTC can support,
and added to a running total of population supportability.
In addition, the earliest LAD of all Harvest Bare kits
programmed for the base is retrieved and stored for later
use.

The program then checks for the presence of both
Harvest Eagle and Harvest Bare kits at the same base. If
both Harvest Eagle and Harvest Bare kits are programmed for
a base, an error message is displayed. The total population
that programmed housekeeping can support is then multiplied
times 1.2 and compared to the base population. If it is
smaller, an error message is displayed.

Combat Support Group. If a Harvest Eagle kit is
planned for a base, the CSG check is performed. The CSG UTC
is retrieved from the CSG file and the number of times it
occurs in the AIR_TPFDD file is added and multiplied by the

40

population it can support. This number is then multiplied by 1.1 and compared to the base population. If it is smaller, an error message is displayed.

Harvest Bare Advance. If a Harvest Bare kit was programmed for a base, a check for a Harvest Bare advance is made. All advance UTCs are checked against the AIR_TPFDD file, and the earliest LAD of these is saved and compared to the Harvest Bare LAD stored earlier. If the Harvest Bare LAD is earlier than the advance LAD, or if no advance at all is planned, an error message is displayed.

MOSS. If a Harvest Bare kit is planned for a base, the flying squadrons need MOSS units. First, a record from the MOSS array is retrieved, and the AIR_TPFDD file is examined for every occurrence of any of the up to three aircraft UTCs. This total is stored. Next, each of the up to four MOSS UTCs on the record is added and that total is multiplied by the number of aircraft squadrons each will support. The total of aircraft squadrons that need MOSS support is compared to the number of squadrons that can be supported by the planned MOSS. If the number of aircraft squadrons MOSS units can support is smaller than the number of aircraft squadron UTCs, an error message is displayed.

Fuel. There must be fuel support for every flying squadron planned for a base. First, each fuel support UTC, from the FUELS array and the number of flying squadrons it will support is processed against the AIR_TPFDD file for

41

number of occurences. The number of occurences is then
multiplied by the number of flying squadrons that particular
fuel UTC can support, and added to a running total of fuel
support. Next, each UTC is read from the FLY_SQN file, and
the number of times it occurs in the AIR_TPFDD file is added
to a running total until all flying squadron UTCs have been
processed. Then, the number of flying squadrons that can be
supported by programmed fuel support is compared to the
total number of flying squadrons planned for that base. If
it is smaller, an error message is displayed.

STAMP and STRAPP. STAMP and STRAPP are processed
in different sub-routines, but both work the same way.
Fighter squadrons require STAMP and STRAPP support. These,
and the STAMP and STRAPP UTCs they need are in the STMP and
STRP files respectively. First, a record is read from the
STMP/STRP file. The UTC for the STAMP or STRAPP is
processed against the AIR_TPFDD file and the number of times
it occurs is totaled. Next, each of the up to four aircraft
UTCs that use that STAMP or STRAPP are processed against the
AIR_TPFDD file and the number of times they occur is
totaled. The total number of flying squadrons needing a
particular STAMP or STRAPP is compared to the total number
of that STAMP or STRAPP programmed, and if it is larger, an
error message is displayed.

UTC Sequencing. There are several types of units
that must arrive in a predetermined sequence. Those units,
in order of required arrival, are; Site Survey, Aerial Port,

42

Red Horse or Prime Beef, Housekeeping, Combat Support Group (if a Harvest Eagle is programmed), and Security. The UTCs for each type of unit are in separate data files, for example, there is a file for Site Survey UTCs called SURVEY, a file for Aerial Port UTCs called A_PORT, etc. The program first processes all UTCs in the SURVEY file against the AIR_TPFDD file for the presence of any of the Site Survey UTCs at the base being processed. The earliest LAD of all Site Survey UTCs programmed for the base is stored. It then extracts the LAD for the other unit types in the sequence in a similar manner. After all of the LADs have been extracted, comparisons are made. The Site Survey LAD is compared to the Aerial Port LAD. If the Site Survey LAD is later, an error message is displayed. The Aerial Port LAD is then compared to the Red Horse/Prime Beef LAD, and so on, until the sequencing for Security has been checked. The arrival sequence is set in the program, and any changes to the sequence will require that the program be modified.

UTC Timing. Figure 3 on page 44 illustrates the sequence of events in JPLAN. D-Day is the day on which hostilities or operations begin. Numbers on the line to the left of D-Day are the number of days before combat operations begin, while numbers to the right of D-Day are days after hostilities begin. Therefore, D-3 is three days before hostilities begin, D-4 is four days before, D+3 is three days after, etc. There are a number of UTCs that must

43

have an LAD prior to D-3. These UTCs are processed in three

separate sub-routines. The UTCs that must arrive in

sequence must also arrive prior to D-3. Those units are

checked for late arrival at the same time the sequence is

checked. The second timing check is done on flying squadron

UTCs. Records are extracted individually from the FLY_SQN

file and processed against the AIR_TPFDD file, checking for

the presence of that UTC at the base being processed. If

the UTC is programmed at the base being processed, and its

LAD is D-3 or less, an error message is displayed. A timing

check for the remaining units where timing is important is

done in a third sub-routine. Those types of units are;

Transient Maintenance, Communications, Fuel Support, Supply

Support, Field Maintenance, and Munitions Maintenance. All

are processed in a manner similar to the aircraft UTCs

except for Field and Munitions Maintenance, which are

aircraft dependent. For those two unit types, a check is

first made for the presence of each particular aircraft UTC

that requires Field or Munitions Maintenance. For every

aircraft UTC that is found, a search is made for its

corresponding Field or Munitions Maintenance unit, and their

LADs compared ensure they are lower than D-3. If they are

not, an error message is displayed.

```
D-10 --------- D-3 ----- D-DAY ----- D+3 --------- D+10
Deployment        Flying   Hostilities
  Begins        Ops Begin    Begin
```

Figure 3. D-Day Time Line

Testing and Debugging. Following the writing and entering of the computer code, testing and debugging began. The first step in debugging is compiling the program. This process uses a computer program which takes the program code, which is in human readable format called source code (See Appendix B), and converts it to computer readable format called object code. While the compiler performs this operation, it also checks the commands in the code for compliance to the rules of the particular computer programming language. If any of the commands are found to be in error, the compiler flags those errors and stops the compile process. These errors are called syntax errors, and must be corrected before the program can be successfully compiled and run.

In this particular program, since there was a mix of C commands and INGRES commands in the same program, two compiles and a linker had to be run to make an executable program. The first compile took INGRES commands and converted them into source code C commands. Once all of the syntax errors were corrected on this compile, the C compile was run. The C compile takes all of the source code and creates the object code. After all of the C syntax errors were corrected, the linker was run. The linker joins the object code with the C and INGRES command files it needs to run and creates the executable file that the user will run by simply typing the file name.

Following error-free compiles, actual program testing began. Four sets of student data files from a previous Combat Logistics class were used in testing. Between all four databases, there were enough different conditions to adequately test the evaluation program. Prior to testing the program with this data, the data was manually evaluated using the same criteria the program would use. This was done so it would be known what to expect when the program was run, and it would be known when the program made an error.

Errors that turned up during this phase of testing were logic errors. A logic error indicates that the syntax of the commands is correct, but the sequence in which they are executed is wrong, or the wrong commands are used. Whatever the cause, when a program has a logic error, the program either does not do what is expected, or it does not complete running at all.

Logic errors are sometimes hard to fix because the programmer does not always know where in the program they are occurring. That was not the case in this program. A number of print messages were put inserted into the program code, so that when the program ran, it left a trail of messages. If something unexpected occurred while the program was running, all that needed to be done was to follow the trail of messages through the source code, and the exact location of a logic error could be easily found.

Following successful evaluations of all of the test data, the program was considered complete.

Program Implementation

Evaluation System Installation. Implementation of the JPLAN evaluation was extremely simple. To install the evaluation system, the user simply follows the JPLAN installation instructions outlined in Chapter 2 of the JPLAN Monitor User's Manual (8:3-4). Evaluation system installation was integrated into the JPLAN installation so that there are no changes in the way the user installs JPLAN.

When JPLAN is installed onto a computer for an exercise, two installation programs copy the contents of the JPLAN floppy disks onto the C: drive (hard disk) of the receiving computer. One installation program copies the JPLAN program and INGRES files, while the other copies the JPLAN database files. Implementation of the evaluation system simply involved incorporating evaluation system files into the existing JPLAN installation procedure. For those familiar with the JPLAN installation procedure before the evaluation system, the only visible difference is that the install program now prompts the user to insert the evaluation system disk in the A: (floppy disk) drive.

In incorporating evaluation system installation into JPLAN installation, the evaluation system program, the batch file that executes it, and a batch program to generate the

47

hardcopy printout of evaluation system output were put on a floppy disk. The batch program that installs JPLAN was modified to tell the user to insert the evaluation system disk after which it would load those three files into the \INGRES\BIN directory.

Next, all of the new data files that were created, along with some INGRES system-created files from the INGRES\DATA\JPLAN directory were copied onto the JPLAN data disks used for installation. A batch program was created for this purpose. When JPLAN data is installed onto a computer for an exercise, the batch program that does the installation of the data files simply copies all files from the data disks to the C: drive of the receiving computer, so it was not necessary to change the JPLAN database installation program. Following JPLAN installation, the evaluation system is ready run.

Program Execution. To begin the JPLAN evaluation, the user only needs to enter EVALJPLN, and hit return. The program will ask the user if Hardcopy output, Screen output, or Both is desired. Entering 'S' to this option will send all evaluation system messages to the CRT screen, with no hardcopy. Entering 'H' will send all evaluation system messages to a file called EVALOUT.TXT. After the program has completed running, entering 'EVALPRNT' will send the output file to the printer. Entering 'B' will give both screen output and file output. Once the user answers the

output format question, there is no further user-program interface.

Another output option is to enter 'CTRL P' on the computer keyboard before program execution. This command makes everything that displays on the CRT echo on the printer. If this is done, enter 'S' or 'B' to the hardcopy option question, and all messages that display on the CRT will also print on the printer at the same time. To stop messages from echoing on the printer, enter 'CTRL P' a second time.

The evaluation system, at the present time, takes about 30-35 minutes to complete its evaluation of the entire JPLAN database. When the program completes its evaluation, a message is displayed saying that the JPLAN evaluation is complete. The program will then terminate.

IV. Recommendations and Conclusions

Recommendations

There are several recommendations for improvement, all which would enhance the evaluation system. First, the program code could probably be made more efficient, which would allow it to run faster. As it is presently written, the evaluation system takes approximately 30-35 minutes to process the present JPLAN database with 6 bases in the AIRBASE file. This task should be taken on by someone that knows C and INGRES so that they can follow the logic of the present code and apply more efficient programming methods to it.

The development of an interactive program to maintain the data files created for the evaluation system would also enhance the evaluation program. While INGRES can be used interactively to maintain these files, it can be intimidating to the first time or infrequent user because of the sheer number of options available in interactive INGRES. This interactive program should allow the user to update just the evaluation system files. It should employ forms, and allow for displaying, adding, deleting, modifying and sorting records in the evaluation system data files.

Conclusions

This thesis has shown that it is possible to computerize the evaluation of student performance in the JPLAN Exercise system. Although the program could be optimized to run faster, at its present speed it still evaluates student performance faster than previous manual methods, and frees the instructor to do other things while the evaluation system is performing its evaluations.

There is room for improvement in the evaluation system; few, if any, computer systems are perfect on their first iteration. The recommendations mentioned above will make the evaluation system a more valuable tool for the Combat Logistics instructors, and as they use the system they will think of other ways to improve it.

Appendix A: Definitions

The following is a list of definitions to terms that will be used:

Aerial Port: Provides freight/passenger services, inspection of air drop loads, and organic maintenance for a contingency air terminal (3:71-74).

Bare Base: A base which has as a minimum, a source of potable water, a runway, taxiways and parking areas adequate for a deployed force (4:57).

C-Day: The unnamed day which a deployment begins, or is scheduled to begin (6:A1-1-1-3)

Combat Support (CS) Forces: Forces whose primary missions are to give combat support (operational assistance) to combat forces (2:11.5). Includes such unit types as maintenance, transportation, and supply.

Combat Service Support (CSS) Forces: Forces whose primary missions are to give service support to combat (and combat support) forces (2:11.5). Includes such unit types as, food services, mortuary affairs, and recreation services.

Concept of Operations: A broad outline of a commander's assumptions or intent in regard to an operation or series of operations. Concept of Operations gives an overall picture of the operation (2:11.6).

D-Day: The unnamed day on which hostilities, an operation or an exercise begins or is scheduled to begin (6:A1-1-1-';).

Database: Collection of one or more computer files that represent all the data associated with, or supporting the objective of an ADP system (2:11.7).

Database Management System: A system used for storing, retrieving and formatting information to and from a database (1:77).

Force List: List of the total combat, CS, and CSS forces required by an OPLAN. Includes assigned forces, augmentation forces and other forces to be employed in support of the plan (2:11.13).

Harvest Bare (HB): A bare base package which consists of modular shelters, utilities, base maintenance equipment and support systems necessary to convert a bare base into an operational base. Harvest Bare packages in JPLAN are able to support up to 4500 personnel in 1500 man increments, and are maintained in a ready-to-deploy condition (3:29). Harvest Bare packages are made up of 3 components: Base Augmentation Support Set (BASS), Maintenance/Operations Support Shelters (MOSS), and Modular Air Transportable Hospital (MATH) (4:63).

Harvest Eagle (HE): Air transportable bare base housekeeping packages capable of supporting up to 1100 personnel. HE kits include tents, administrative equipment, chaplain supplies, cots, sleeping bags, heaters, water purifiers, electrical generators and kitchen equipment. HE kits are not weapon-system-specific (4:64).

Joint Operation Planning System (JOPS): DOD directed, JCS specified system for planning regional and global, joint military operations. Does not include strategic plans (SIOP) (2:11.17).

JOPS ADP: WWMCCS computer-based system to support JOPS (2:11.19).

Latest Arrival Date (LAD): Latest day, relative to C-Day in which a unit can arrive at its port of debarkation, and still support the OPLAN (2:11.19).

Limiting Factor (LIMFAC): A factor or condition that impedes mission accomplishment (2:11.19).

Maintenance/Operations Support Shelters (MOSS): That part of a HB kit that contains reusable, expandable shelters that are weapon system peculiar (4:63).

Modular Air Transportable Hospital (MATH): Medical support portion of a HB package. Consists of medical equipment and modules (4:63).

Port of Debarkation (POD): The location to which a unit will deploy. May or may not be the final destination (2:11.25).

Port of Embarkation (POE): Location from which a movement of a unit begins. May or may not be the point of origin (2:11.26).

Short Ton (STON): Unit of weight measure (2000lbs) (2:11.28).

Shortfall: A lack of forces, equipment, personnel, or capability that affects the ability to accomplish a mission (2:11.28).

Standard Air Munitions Package (STAMP): A weapon-system-specific deployable munitions package designed to provide supplies of munitions until resupply can occur (4:68).

Standard Tanks, Racks, Adapters, and Pylons Package (STRAPP): A weapon-system-specific package of tanks, racks, adapters and pylons. Similar to the same service that STAMP does for munitions. (4:68).

Support Forces: Forces whose primary missions are to give combat support or services support to combat forces (2:11.5).

Time-Phased Force Deployment Data (TPFDD): Computer file containing force data of an OPLAN (2:11.31). Includes UTC, a short unit description, number of personnel, number of tons, FAD, and LAD.

Type Unit Data File (TUCHA): A computer file containing planning data and movement characteristics for personnel, cargo and supplies (2:11.34).

Unified Command: A command which consists of the forces from two or more services, and whose purpose is to deploy and employ US military forces in the most effective way (2:2.12).

Unit Type Code (UTC): A five position code that communicates force requirement information to the Joint Operation Planning System (JOPS). They can provide information about either in-place or deploying units (3:2),(2:11.34).

Appendix B: Evaluation System Program Code

Following is the source program code for the JPLAN
Evaluation System:

```
/**********************************************************
*                                                        *
*           JPLAN SIMULATION EVALUATION                  *
*                                                        *
*              by Capt Chip Jean                         *
*                                                        *
*              AFIT  Class 88-S                          *
*                                                        *
**********************************************************/


/**********************************************************
*          Declare INGRES Tables Used in Program         '
**********************************************************/

EXEC SQL INCLUDE sqlca;

EXEC SQL DECLARE   tucha TABLE
      (utc              vchar(5),
       des              vchar(16),
       pers             integer2,
       bpers            integer2,
       pax              integer2,
       stons            integer2,
       osize            integer2,
       non_air_tran     integer2,
       svc              vchar(2));

EXEC SQL DECLARE air_tpfdd TABLE
      (line_number      integer2,
       utc              vchar(5),
       ab_pod           vchar(3),
       emd              integer2,
       lad              integer2,
       pri              integer2);

EXEC SQL DECLARE airbase TABLE
      (ab_pod           vchar(3),
       base_name        vchar(12),
       pax_cap          integer4,
       ab_cargo_cap     integer4,
       max_ramp_sp      integer4,
```

```
EXEC SQL DECLARE m10pod TABLE
      (lad            integer2,
       pod            vchar(3),
       pax            integer2,
       stons          integer2,
       osize          integer2,
       army           integer2,
       usaf           integer2,
       bpers          integer2);

EXEC SQL DECLARE m10tot TABLE
      (lad            integer2,
       pax            integer2,
       stons          integer2,
       osize          integer2,
       army           integer2,
       usaf           integer2,
       bpers          integer2);

EXEC SQL DECLARE aerial_port TABLE
      (utc            vchar(5),
       ap_pax_cap     integer2,
       ap_cargo_cap   integer2);

EXEC SQL DECLARE stmp TABLE
      (stmp_utc       vchar(5),
       sta_ac1        vchar(5),
       sta_ac2        vchar(5),
       sta_ac3        vchar(5),
       sta_ac4        vchar(5));

EXEC SQL DECLARE strp TABLE
      (strp_utc       vchar(5),
       str_ac1        vchar(5),
       str_ac2        vchar(5),
       str_ac3        vchar(5),
       str_ac4        vchar(5));

EXEC SQL DECLARE eagle TABLE
      (eagle_utc      vchar(5),
       eagle_pop      integer2);

EXEC SQL DECLARE bare TABLE
      (bare_utc       vchar(5),
       bare_spt       integer2);

EXEC SQL DECLARE supply TABLE
      (supply_utc     vchar(5),
       supply_spt     integer2);

EXEC SQL DECLARE fuels TABLE
      (fuels_utc      vchar(5),
       fuels_sqn      integer2);
```

```
EXEC SQL DECLARE csg TABLE
     (csg_utc        vchar(5),
      csg_spt        integer2);

EXEC SQL DECLARE a_port TABLE
     (port_utc       vchar(5),
      port_tons      integer2);

EXEC SQL DECLARE t_maint TABLE
     (tm_utc         vchar(5),
      tm_spt         integer2);

EXEC SQL DECLARE math TABLE
     (math_utc       vchar(5));

EXEC SQL DECLARE army TABLE
     (grunts         integer2);

EXEC SQL DECLARE adv TABLE
     (hb_adv_utc     vchar(5));

EXEC SQL DECLARE hb_moss TABLE
     (moac1          vchar(5),
      moac2          vchar(5),
      moac3          vchar(5),
      moutc1         vchar(5),
      mono1          integer2,
      moutc2         vchar(5),
      mono2          integer2,
      moutc3         vchar(5),
      mono3          integer2,
      moutc4         vchar(5),
      mono4          integer2);

EXEC SQL DECLARE fly_sqn TABLE
     (fly_utc        vchar(5));

EXEC SQL DECLARE survey TABLE
     (survey_utc     vchar(5));

EXEC SQL DECLARE beef TABLE
     (beef_utc       vchar(5));

EXEC SQL DECLARE security TABLE
     (sp_utc         vchar(5));

EXEC SQL DECLARE comm TABLE
     (comm_utc       vchar(5));

EXEC SQL DECLARE fmaint TABLE
     (ftype          vchar(5),
      fac_utc1       vchar(5),
```

```
        fac_utc2          vchar(5),
        fac_utc3          vchar(5),
        fmutc1            vchar(5),
        fmutc2            vchar(5),
        fmutc3            vchar(5));

EXEC SQL DECLARE mmaint TABLE
        (mtype            vchar(5),
        mac_utc1          vchar(5),
        mac_utc2          vchar(5),
        mac_utc3          vchar(5),
        mmutc1            vchar(5),
        mmutc2            vchar(5),
        mmutc3            vchar(5));


/************************************************************
*                    DECLARE STRUCTURES                    *
************************************************************/

EXEC SQL BEGIN DECLARE SECTION;

        struct fldtab_{
                char      fldtype[6];
                char      fldac1[6];
                char      fldac2[6];
                char      fldac3[6];
                char      fldutc1[6];
                char      fldutc2[6];
                char      fldutc3[6];
                } fldtab[20];

        struct muntab_{
                char      muntype[6];
                char      munac1[6];
                char      munac2[6];
                char      munac3[6];
                char      munutc1[6];
                char      munutc2[6];
                char      munutc3[6];
                } muntab[20];

        struct motab_ {
                char      moa1[6];
                char      moa2[6];
                char      moa3[6];
                char      mou1[6];
                short     mou1n;
                char      mou2[6];
                short     mou2n;
                char      mou3[6];
                short     mou3n;
                char      mou4[6];
```

58

```
        short       mou4n;
        }   motab[20];

struct base_tab_  {
        char        base_pod[4];
        char        base_name[13];
        long        base_cap;
        } base_tab[20];

 struct he_tab_  {
        char        he_utc[6];
        short       he_pop;
        } he_tab[20];

 struct hb_tab_  {
        char        hb_utc[6];
        short       hb_pop;
        } hb_tab[20];

 struct supp_tab_  {
        char        supply_utc[6];
        short       pop_spt;
        } supp_tab[20];

 struct fuel_tab_  {
        char        fuel_utc[6];
        short       fuel_sqn;
        } fuel_tab[20];

 struct csg_tab_  {
        char        cs_utc[6];
        short       cs_spt;
        } csg_tab[20];

 struct ap_tab_  {
        char        ap_utc[6];
        short       ap_spt;
        } ap_tab[20];

 struct maint_tab_  {
        char        maint_utc[6];
        short       maint_spt;
        } maint_tab[20];

 struct m10_tab_  {
        short       tons;
        short       bpop;
        }m10_tab[20];

 char ma_utc[20][6];
 char adv_utc[20][6];
 char fly_sqn[20][6];
```

```
/***********************************************************************
*                      DECLARE VARIABLES                               *
***********************************************************************/

        int base_idx = 0;      /* Index for Base Table */
        int base_max = 0;      /* # Entries for Base Table */
        int he_idx = 0;        /* Index for Harvest Eagle Table */
        int he_max = 0;        /* # Entries for Harvest Eagle Table */
        int hb_idx = 0;        /* Index for Harvest Bare Table */
        int hb_max = 0;        /* # Entries for Harvest Bare Table */
        int supp_idx = 0;      /* Index for Supply Table */
        int supp_max = 0;      /* # Entries for Supply Table */
        int fuel_idx = 0;      /* Index for Fuel Table
        int fuel_max = 0;      /* # Entries for Fuel Table */
        int csg_idx = 0;       /* Index for CSG Table */
        int csg_max = 0;       /* # Entries for Combat Spt Grp Table */
        int ap_idx = 0;        /* Index for Aerial Port Table */
        int ap_max = 0;        /* # Entries for Aerial Port Table */
        int maint_idx = 0;     /* Index for Transient Maint Table*/
        int maint_max = 0;     /* # Entries for Maintenance Table */
        int math_idx = 0;      /* Index for MATH Table */
        int math_max = 0;      /* # Entries for Hospital Table */
        int count_1;           /* Used to hold # of occurences */
        int avg_tons = 0;      /* Avg tons a base can handle daily */
        int tot_pop = 0;       /* Total population at a base */
        int army_pop = 0;      /* Next 3 used in checking Army pop */
        int army_tot = 0;
        int tot_grunt = 0;
        char hold_utc[6];      /* Temp holding area for UTCs */
        int hold_tot = 0;      /* Used to keep running totals */
        int he_chk = 0;        /* 0 if no HE, 1 if HE found */
        int hb_chk = 0;        /* 0 if no HB, 1 if HB found */
        int bare_lad = 0;      /* Latest arrival date for HB */
        int bcomp_lad = 10;    /* Used for comparing HB lad */
        int adv_lad = 0;       /* HB advance LAD */
        int acomp_lad = 10;    /* Used for comparing advance LAD */
        int adv_idx = 0;       /* Index for adv table */
        int adv_max = 0;       /* # entries for advance table */
        int sqn_idx = 0;       /* Index for fly sqn table */
        int sqn_max = 0;       /* # entries for flying UTCs */
        int moidx = 0;         /* Index for MOSS table */
        int momax = 0;         /* # entries for MOSS table */
        int hold_it = 0;       /* next 2, holding areas */
        int hld_cnt = 0;
        char outform;          /* Holds output form request */
        int bare_pop;
        int form_ok = 0;
        int hsb = 0;
        int survey_lad = 11;   /* Site Survey LAD */
        int ap_lad = 11;       /* Aerial Port LAD */
        int hk_lad = 11;       /* Housekeeping LAD */
        int csg_lad = 11;      /* CSG LAD */
```

```c
int sp_lad = 11;      /* Security LAD
int beef_lad = 11;    /* Prime Beef/Red Horse LAD */
int needs_fuel = 0;   /* 1=needs fuel 0=does not need fuel */
int munidx = 0;       /* Index for munitions maint table */
int munmax = 0;       /* # entries in mun maint table */
int fldidx = 0;       /* Index for field maint table */
int fldmax = 0;       /* # entries for field maint table */

EXEC SQL END DECLARE SECTION;


/*********************************************************
 *                  START MAIN PROGRAM                   *
 *********************************************************/

#include <c:\msc\include\signal.h>
#include <c:\msc\include\stdio.h>
#include <c:\msc\include\ctype.h>
#include <c:\msc\include\math.h>
#include <c:\msc\include\process.h>

FILE *eout;
main ()
{

EXEC SQL WHENEVER sqlerror STOP;
printf (" \n\n\n\n\n");
printf ("                        WELCOME TO THE\n");
printf ("                    JPLAN Simulation Evaluation\n\n\n");
printf ("        This Program Will Evaluate the JPLAN"
                "Database\n");
printf ("For Supportability Shortfalls and Timing and"
                "Sequence Errors\n\n");
printf ("NOTE:  If you enter H or B to the following"
                "question, you must\n");
printf ("        enter EVALPRNT at the C: prompt after this"
                "program is\n");
printf ("        complete, to get your hardcopy output."

                "\n\n");
do
  {
  if (count_1 > 0) printf ("ERROR - Entry Must Be H, S, or"
                " B\n\n");
  count_1 = 1;
  printf ("Select Output Media - Hardcopy, Screen, or "
                "Both\n\n");
  printf ("Enter  H  for Hardcopy Output Only\n");
  printf ("Enter  S  for Screen Output Only\n");
  printf ("Enter  B  for Both Hardcopy and Screen Output"
                "\n\n");
  outform = getch();
  if ((outform == 'H') || (outform == 'h')) hsb = 3;
```

```
     if ((outform == 'S') !! (outform == 's')) hsb = 1;
     if ((outform == 'B') !! (outform == 'b')) hsb = 2;
     }
while (hsb < 1);
if (hsb > 1)
     eout = fopen("evalout.txt","w");
bld_base ();
bld_adv ();
bld_he ();
bld_hb ();
bld_supply ();
bld_fuel ();
bld_csg ();
bld_maint ();
bld_ap ();
bld_math ();
bld_sqn ();
bld_moss ();
bld_fld ();
bld_mun ();
army ();
medical ();
for (base_idx = 0; base_idx < base_max; base_idx ++)
   {
   he_chk = 0; hb_chk = 0; count_1 = 0;
   if (hsb < 3)
      {
      printf (" \n");
      printf ("**************************************************\n");
      printf ("               EVALUATING BASE %s \n",
           base_tab[base_idx].base_name);
      printf ("**************************************************\n\n");
      }
   if (hsb > 1)
      {
      fprintf (eout," \n");
      fprintf (eout,"**************************************************\n");
      fprintf (eout,"               EVALUATING BASE %s \n",
           base_tab[base_idx].base_name);
      fprintf (eout,"**************************************************\n\n");
      }
      base_tots ();
      supply ();
      maint ();
      tons ();
      he_hb ();
      if (he_chk > 0) csg ();
      if (hb_chk > 0) moss ();
      chk_stamp ();
      chk_strapp ();
      fuels ();
      sequence ();
      acft_timing ();
```

```c
      misc_timing ();
   }
if (hsb < 3)
   {
   printf ("**********************************************\n");
   printf ("*        JPLAN EVALUATION COMPLETE        *\n");
   printf ("**********************************************\n");
   }
if (hsb > 1)
   {
   fprintf (eout,"**********************************************\n");
   fprintf (eout,"*     JPLAN EVALUATION COMPLETE        *\n");
   fprintf (eout,"**********************************************\n");
   fclose(eout);
   }
EXEC SQL DISCONNECT;
}


/**********************************************************************
*                      BUILD BASE TABLE                              *
**********************************************************************/

bld_base ()
{

if (hsb < 3)
   {
   printf ("\n\n");
   printf ("------------BUILDING TABLES------------\n\n");
   }
EXEC SQL CONNECT jplan;
EXEC SQL DECLARE base_csr CURSOR FOR
      select distinct
            ab_pod,
            base_name,
            ab_cargo_cap
      from airbase
      order by ab_pod;
EXEC SQL OPEN base_csr;
EXEC SQL FETCH base_csr into :base_tab[base_idx];
while (sqlca.sqlcode == 0)
      {
      base_idx ++;
      base_max ++;
      EXEC SQL FETCH base_csr into :base_tab[base_idx];
      }
EXEC SQL CLOSE base_csr;
}
```

```
/***********************************************************
*              BUILD HARVEST BARE ADVANCE TABLE            *
***********************************************************/

bld_adv ()
{

EXEC SQL DECLARE adv_csr CURSOR FOR
     select distinct
          hb_adv_utc
     from adv;
EXEC SQL OPEN adv_csr;
EXEC SQL FETCH adv_csr into :adv_utc[adv_idx];
while (sqlca.sqlcode == 0)
     {
     adv_idx ++;
     adv_max ++;
     EXEC SQL FETCH adv_csr into :adv_utc[adv_idx];
     }
EXEC SQL CLOSE adv_csr;
}


/***********************************************************
*                 BUILD HARVEST BARE TABLE                *
***********************************************************/

bld_hb ()
{

EXEC SQL DECLARE hb_csr CURSOR FOR
     select distinct
          bare_utc,
          bare_spt
     from bare;
EXEC SQL OPEN hb_csr;
EXEC SQL FETCH hb_csr into :hb_tab[hb_idx];
while (sqlca.sqlcode == 0)
     {
     hb_idx ++;
     hb_max ++;
     EXEC SQL FETCH  hb_csr into :hb_tab[hb_idx];
     }
EXEC SQL CLOSE hb_csr;
}
```

```
/**********************************************************
*                 BUILD HARVEST EAGLE TABLE               *
**********************************************************/

bld_he ()
{

EXEC SQL DECLARE he_csr CURSOR FOR
     select distinct
          eagle_utc,
          eagle_pop
     from eagle;
EXEC SQL OPEN he_csr;
EXEC SQL FETCH he_csr into :he_tab[he_idx];
while (sqlca.sqlcode == 0)
     {
     he_idx ++;
     he_max ++;
     EXEC SQL FETCH he_csr into :he_tab[he_idx];
     }
EXEC SQL CLOSE he_csr;
}


/**********************************************************
*                 BUILD SUPPLY TABLE                      *
**********************************************************/

bld_supply ()
{

EXEC SQL DECLARE supp_csr CURSOR FOR
     select distinct
          supply_utc,
          supply_spt
     from supply;
EXEC SQL OPEN supp_csr;
EXEC SQL FETCH supp_csr into :supp_tab[supp_idx];
while (sqlca.sqlcode == 0)
     {
     supp_idx ++;
     supp_max ++;
     EXEC SQL FETCH supp_csr into :supp_tab[supp_idx];
     }
EXEC SQL CLOSE supp_csr;
}
```

```
/***********************************************************
*                    BUILD FUELS TABLE                     *
***********************************************************/

bld_fuel ()
{

EXEC SQL DECLARE fuel_csr CURSOR FOR
     select distinct
            fuels_utc,
            fuels_sqn
     from fuels;
EXEC SQL OPEN fuel_csr;
EXEC SQL FETCH fuel_csr into :fuel_tab[fuel_idx];
while (sqlca.sqlcode == 0)
     {
     fuel_idx ++;
     fuel_max ++;
     EXEC SQL FETCH fuel_csr into :fuel_tab[fuel_idx];
     }
EXEC SQL CLOSE fuel_csr;
}


/***********************************************************
*          BUILD COMBAT SUPPORT GROUP TABLE               *
***********************************************************/

bld_csg ()
{

EXEC SQL DECLARE csg_csr CURSOR FOR
     select distinct
            csg_utc,
            csg_spt
     from csg;
EXEC SQL OPEN csg_csr;
EXEC SQL FETCH csg_csr into :csg_tab[csg_idx];
while (sqlca.sqlcode == 0)
     {
     csg_idx ++;
     csg_max ++;
     EXEC SQL FETCH csg_csr into :csg_tab[csg_idx];
     }
EXEC SQL CLOSE csg_csr;
}
```

```
/*********************************************************
*                 BUILD MAINTENANCE TABLE               *
*********************************************************/

bld_maint ()
{

EXEC SQL DECLARE maint_csr CURSOR FOR
     select distinct
            tm_utc,
            tm_spt
     from t_maint;
EXEC SQL OPEN maint_csr;
EXEC SQL FETCH maint_csr into :maint_tab[maint_idx];
while (sqlca.sqlcode == 0)
     {
     maint_idx ++;
     maint_max ++;
     EXEC SQL FETCH maint_csr into :maint_tab[maint_idx];
     }
EXEC SQL CLOSE maint_csr;
}



/*********************************************************
*                 BUILD AERIAL PORT TABLE               *
*********************************************************/

bld_ap ()
{

EXEC SQL DECLARE ap_csr CURSOR FOR
     select distinct
            port_utc,
            port_tons
     from a_port;
EXEC SQL OPEN ap_csr;
EXEC SQL FETCH ap_csr into :ap_tab[ap_idx];
while (sqlca.sqlcode == 0)
     {
     ap_idx ++;
     ap_max ++;
     EXEC SQL FETCH ap_csr into :ap_tab[ap_idx];
     }
EXEC SQL CLOSE ap_csr;
}

/*********************************************************
*                 BUILD HOSPITAL TABLE                  *
*********************************************************/

bld_math ()
{
```

```
EXEC SQL DECLARE math_csr CURSOR FOR
     select distinct
          math_utc
     from math;
EXEC SQL OPEN math_csr;
EXEC SQL FETCH math_csr into :ma_utc[math_idx];
while (sqlca.sqlcode == 0)
     {
     math_idx ++;
     math_max ++;
     EXEC SQL FETCH math_csr into :ma_utc[math_idx];
     }
EXEC SQL CLOSE math_csr;
}

/***********************************************************
*                    BUILD FLYING SQUADRON TABLE           *
***********************************************************/

bld_sqn ()
{

EXEC SQL DECLARE bld_sqn_csr CURSOR FOR
     select
          fly_utc
     from fly_sqn;
EXEC SQL OPEN bld_sqn_csr;
EXEC SQL FETCH bld_sqn_csr into :fly_sqn[sqn_idx];
while (sqlca.sqlcode == 0)
     {
     sqn_idx ++;
     sqn_max ++;
     EXEC SQL FETCH bld_sqn_csr into :fly_sqn[sqn_idx];
     }
EXEC SQL CLOSE bld_sqn_csr;
}

/***********************************************************
*                    BUILD MOSS TABLE                      *
***********************************************************/

bld_moss ()
{

EXEC SQL DECLARE moss_csr CURSOR FOR
     select
          moac1, moac2, moac3, moutc1, mono1, moutc2, mono2,
          moutc3, mono3,moutc4, mono4
     from hb_moss;
EXEC SQL OPEN moss_csr;
while (sqlca.sqlcode == 0)
     {
```

```
        EXEC SQL FETCH moss_csr into :motab[moidx];
        moidx ++;
        momax ++;
        }
EXEC SQL CLOSE moss_csr;
}




/************************************************************
*               BUILD FIELD MAINTENANCE TABLE              *
************************************************************/

bld_fld ()
{

EXEC SQL DECLARE bldfld_csr CURSOR FOR
        select
              ftype, fac_utc1, fac_utc2, fac_utc3, fmutc1,
               fmutc2, fmutc3
        from fmaint;
EXEC SQL OPEN bldfld_csr;
while (sqlca.sqlcode == 0)
        {
        EXEC SQL FETCH bldfld_csr into :fldtab[fldidx];
        fldidx ++;
        fldmax ++;
        }
EXEC SQL CLOSE bldfld_csr;
}




/************************************************************
*           BUILD MUNITIONS MAINTENANCE TABLE             *
************************************************************/

bld_mun ()
{

EXEC SQL DECLARE bldmun_csr CURSOR FOR
        select
              mtype, mac_utc1, mac_utc2, mac_utc3, mmutc1,
                mmutc2, mmutc3
        from mmaint;
EXEC SQL OPEN bldmun_csr;
while (sqlca.sqlcode == 0)
        {
        EXEC SQL FETCH bldmun_csr into :muntab[munidx];
        munidx ++;
        munmax ++;
        }
EXEC SQL CLOSE bldmun_csr;
}
```

69

```
/***********************************************************
*              DONE BUILDING TABLES                       *
***********************************************************
*              BEGIN REAL PROCESSING                      *
***********************************************************
*         CHECK ARMY POPULATION AGAINST PRELOAD           *
***********************************************************/

army ()
{

if (hsb < 3)
   printf ("------------         CHECKING ARMY PRELOAD        "
          "  ------------\n\n");
if (hsb > 1)
   fprintf (eout,"------------   CHECKING ARMY PRELOAD    "
          "------------\n\n");
/*   Compute current Army population totals in database */
EXEC SQL DECLARE army_csr CURSOR FOR
     select
           army
     from m10pod;
EXEC SQL OPEN army_csr;
while (sqlca.sqlcode == 0)
     {
     EXEC SQL FETCH army_csr into :army_pop;
     army_tot = army_tot + army_pop;
     }
army_tot = army_tot - army_pop;
EXEC SQL CLOSE army_csr;
/******   Get Army preload population total    ******/
EXEC SQL DECLARE grunt_csr CURSOR FOR
     select distinct
           grunts
     from army;
EXEC SQL OPEN grunt_csr;
/* Get Army Preload Population */
EXEC SQL FETCH grunt_csr into :tot_grunt;
if (hsb < 3)
     printf ("Army Preload is %d,  Current Army Population"
             " is %d\n",
             tot_grunt, army_tot);
 if (hsb > 1)
     fprintf (eout,"Army Preload is %d,  Current Army"
             " Population is %d\n",tot_grunt, army_tot);
if (hsb < 3)
  if (army_tot < tot_grunt)
     printf ("ERROR - Ending Army Population Less Than"
             " Beginning - %d vs %d\n\n",
             army_tot,tot_grunt);
   else printf ("Army Population OK\n\n");
if (hsb > 1)
```

```
        if (army_tot < tot_grunt)
            fprintf (eout,"ERROR - Ending Army Population Less "
                        "Than Beginning - %d vs %d\n\n",
                        army_tot, tot_grunt);
        else fprintf (eout,"Army Population OK\n\n");
EXEC SQL CLOSE grunt_csr;
}


/*********************************************************
*            MAKE SURE TWO FIELD HOSPITALS ARE INCLUDED        *
*********************************************************/

medical ()
{
if (hsb < 3)
    printf ("------------     CHECKING FOR TWO HOSPITALS     "
            " ------------\n\n");
if (hsb > 1)
    fprintf (eout,"------------ CHECKING FOR TWO HOSPITALS "
                "------------\n\n");
count_1 = 0;
EXEC SQL DECLARE med_csr CURSOR FOR
        select
                utc
        from air_tpfdd
        where utc = :ma_utc[math_idx];
for (math_idx = 0; math_idx < math_max; math_idx ++)
        {
        EXEC SQL OPEN med_csr;
        while (sqlca.sqlcode == 0)
                {
                EXEC SQL FETCH med_csr into :hold_utc;
                count_1 ++;
                }
        count_1 --;
        EXEC SQL CLOSE med_csr;
        }
if (hsb < 3)
    · printf ("Number of Hospitals Programmed is %d\n",
                count_1);
  if (hsb > 1)
        fprintf (eout,"Number of Hospitals Programmed is %d\n",
                count_1);
if (hsb < 3)
  if (count_1 <  2)
        printf ("ERROR - Less Than Two Field Hospitals "
                "Programmed\n\n");
    else printf ("Field Hospitals OK\n\n");
if (hsb > 1)
  if (count_1 <  2)
        fprintf (eout,"ERROR - Less Than Two Field Hospitals"
                " Programmed\n\n");
```

71

```c
        else fprintf (eout."Field Hospitals OK\n\n");
}


/**************************************************************
*       COMPUTE BASE TONNAGE CAPABILITIES AND POPULATION      *
***************************************************************/

base_tots ()
{

avg_tons = 0; tot_pop = 0; count_1 = 0; hold_tot = 0;
hld_cnt = 0;
EXEC SQL DECLARE m10_csr CURSOR FOR
        select
                stons,
                bpers
        from m10pod
        where pod = :base_tab[base_idx].base_pod and
                lad < 1;
EXEC SQL OPEN m10_csr;
while (sqlca.sqlcode == 0)
        {
        EXEC SQL FETCH m10_csr into :hold_tot, :hld_cnt;
        count_1 ++;
        avg_tons = avg_tons + hold_tot;
        tot_pop = tot_pop + hld_cnt;
        }
count_1 --;
tot_pop = tot_pop - hld_cnt;
avg_tons = (avg_tons - hold_tot) / count_1;
if (hsb < 3)
        {
        printf ("Base Population = %d\n\n",tot_pop);
        printf ("Average Daily Tonnage = %d\n\n",avg_tons);
        }
  if (hsb > 1)
        {
        fprintf (eout,"Base Population = %d\n\n",tot_pop);
        fprintf (eout,"Average Daily Tonnage = %d\n\n",
                avg_tons);
        }
EXEC SQL CLOSE m10_csr;
}

/**************************************************************
*       CHECK SUPPLY SUPPORT BASED ON BASE POPULATION         *
***************************************************************/

supply ()
{

hold_tot = 0;
```

```c
if (hsb < 3)
    printf ("------------        EVALUATING SUPPLY SUPPORT      "
                "    -----------\n\n");
  if (hsb > 1)
    fprintf (eout,"------------EVALUATING SUPPLY SUPPORT"
                "------------\n\n");
for (supp_idx = 0; supp_idx < supp_max; supp_idx ++)
    {
    count_1 = 0;
    EXEC SQL SELECT count(utc)
            into :count_1
            from air_tpfdd
            where utc = :supp_tab[supp_idx].supply_utc and
                    ab_pod = :base_tab[base_idx].base_pod;
    hold_tot = hold_tot + (count_1 *
            supp_tab[supp_idx].pop_spt);
    }
hold_tot = hold_tot * 1.1;  /***   10% FUDGE FACTOR   ***/
if (hsb < 3)
    printf ("Supply Can Support a Population of %d\n",
                hold_tot);
if (hsb > 1)
    fprintf (eout,"Supply Can Support a Population of %d\n"
                , hold_tot);
if (hsb < 3)
    if (tot_pop > hold_tot)
      printf ("ERROR - Insufficient Supply Support\n\n");
    else printf ("Supply Support OK\n\n");
if (hsb > 1)
    if (tot_pop > hold_tot)
      fprintf (eout,"ERROR - Insufficient Supply "
                "Support\n\n");
    else fprintf (eout,"Supply Support OK\n\n");
}


/*******************************************************************
*           CHECK TRANSIENT MAINTENANCE SUPPORT              *
*                 BASED ON BASE POPULATION                   *
*******************************************************************/

maint ()
{

hold_tot = 0;
if (hsb < 3)
    printf ("------------ EVALUATING TRANS. MAINT. SUPPORT"
                " -----------\n\n");
  if (hsb > 1)
    fprintf (eout,"------------EVALUATING TRANS. MAINT."
                " SUPPORT------------"
    "\n\n");
for (maint_idx = 0; maint_idx < maint_max; maint_idx ++)
```

73

```
                {
                count_1 = 0;
                EXEC SQL SELECT count(utc)
                        into :count_1
                        from air_tpfdd
                        where utc = :maint_tab[maint_idx].maint_utc and
                                ab_pod = :base_tab[base_idx].base_pod;
                hold_tot = hold_tot + (count_1 *
                        maint_tab[maint_idx].maint_spt);
                }
        hold_tot = hold_tot * 1.1;    /*** 10% FUDGE FACTOR ***/
        if (hsb < 3)
                printf ("Trans. Maint. Can Support a Population of"
                                " %d\n", hold_tot);
        if (hsb > 1)
                fprintf (eout,"Trans. Maint. Can Support a Population "
                                "of %d\n", hold_tot);
        if (hsb > 3)
                if (tot_pop > hold_tot)
                  fprintf (eout,"ERROR - Insufficient Transient "
                                "Maintenance Support\n\n");
                else fprintf (eout,"Transient Maintenance Support OK"
                                "\n\n");
        if (hsb < 3)
                if (tot_pop > hold_tot)
                  printf ("ERROR - Insufficient Transient Maintenance"
                                " Support\n\n");
                else printf ("Transient Maintenance Support OK\n\n");
        }


/******************************************************************
               ...  SUPPORTING CAPABILITY BASED ON THE AVERAGE DAILY*
*AMOUNT THE BASE CAN HANDLE, AND THE AMOUNT OF AERIAL PORT *
*               PROGRAMMED INTO THE BASE                          *
********************************************************************/

tons ()
{

hold_tot = 0;
if (hsb < 3)
        printf ("------------ EVALUATING AERIAL PORT CAPABILITY"
                        " ------------\n\n");
if (hsb > 1)
        fprintf (eout,"------------ EVALUATING AERIAL PORT "
                        "CAPABILITY --------"
                        "----\n\n");
for (ap_idx = 0; ap_idx < ap_max; ap_idx ++)
        {
        count_1 = 0;
        EXEC SQL SELECT count(utc)
                into :count_1
```

```
                from air_tpfdd
                where utc = :ap_tab[ap_idx].ap_utc and
                      ab_pod = :base_tab[base_idx].base_pod;
          hold_tot = hold_tot + (count_1 * ap_tab[ap_idx].ap_spt);
          }
hold_tot = hold_tot * 1.25;     /****25% FUDGE FACTOR***/
if (hsb < 3)
      printf ("Aerial Port Capability is %d\n", hold_tot);
  if (hsb > 1)
      fprintf (eout,"Aerial Port Capability is %d\n",
                  hold_tot);
if (hsb > 1)
      if (avg_tons > hold_tot)
        fprintf (eout,"ERROR - Insufficient Aerial Port"
                  " Support\n\n");
      else fprintf (eout,"Aerial Ports OK\n\n");
if (hsb < 3)
      if (avg_tons > hold_tot)
        printf ("ERROR - Insufficient Aerial Port Support"
                  "\n\n");
      else printf ("Aerial Ports OK\n\n");
}


/*************************************************************
*          CHECK IF HARVEST EAGLE CSG WAS PROGRAMMED         *
*************************************************************/

csg ()
{

hold_tot = 0;
if (hsb < 3)
      printf ("-------------      COMBAT SUPPORT GROUP CHECK "
                  "      -------------\n\n");
if (hsb > 1)
      fprintf (eout,"-------------      COMBAT SUPPORT GROUP "
                  "CHECK      -------------\n\n");
for (csg_idx = 0; csg_idx < csg_max; csg_idx ++)
      {
      count_1 = 0;
      EXEC SQL SELECT count(utc)
            into :count_1
            from air_tpfdd
            where utc = :csg_tab[csg_idx].cs_utc and
                  ab_pod = :base_tab[base_idx].base_pod;
      hold_tot = hold_tot + (count_1 *
            csg_tab[csg_idx].cs_spt);
      }
hold_tot = hold_tot * 1.1;
if (hsb < 3)
      printf ("Population Supported by CSG is %d\n",hold_tot);
if (hsb > 1)
```

```
                fprintf (eout,"Population Supported by CSG is %d\n",
                        hold_tot);
    if (hsb > 1)
            if (tot_pop > hold_tot)
                fprintf (eout,"ERROR - Insufficient CSG for HE\n\n");
            else fprintf (eout,"CSG OK\n\n");
    if (hsb < 3)
            if (tot_pop > hold_tot)
                printf ("ERROR - Insufficient CSG for HE\n\n");
            else printf ("CSG OK\n\n");
    }




/***********************************************************************
 *      CHECK BARE BASE SUPPORT VS PROGRAMMED POPULATION         *
 *      ALSO CHECK FOR BOTH HARVEST BARE AND HARVEST AT THE      *
 *              SAME BASE, WHICH ISN'T ALLOWED                   *
 ***********************************************************************/


he_hb ()
{

bare_pop = 0; hold_tot = 0;
if (hsb < 3)
        printf ("------------        HOUSEKEEPING SUPPORT CHECK    "
                "  ------------\n\n");
if (hsb > 1)
        fprintf (eout,"------------        HOUSEKEEPING SUPPORT"
                        "CHECK     ------------\n\n");
for (he_idx = 0; he_idx < he_max; he_idx ++)
        {
        count_1 = 0;
        EXEC SQL SELECT count(utc)
                into :count_1
                from air_tpfdd
                where utc = :he_tab[he_idx].he_utc and
                        ab_pod = :base_tab[base_idx].base_pod;
        hold_tot = hold_tot + (count_1 *
                he_tab[he_idx].he_pop);
        if (count_1 > 0) he_chk = 1;
        }
if (he_chk > 0) printf ("Harvest Eagle Programmed\n");
/***********************************************************************
 *                   BEGIN HARVEST BARE CHECK                   *
 ***********************************************************************/


EXEC SQL DECLARE bare_csr CURSOR FOR
        select
                utc,
                lad
        from air_tpfdd
        where ab_pod = :base_tab[base_idx].base_pod and
```

```
                    utc = :hb_tab[hb_idx].hb_utc;
acomp_lad = 10; bcomp_lad = 10; bare_lad = 10;
for (hb_idx = 0; hb_idx < hb_max; hb_idx ++)
      {
      count_1 = 0;
      EXEC SQL OPEN bare_csr;
      while (sqlca.sqlcode == 0)
            {
            EXEC SQL FETCH bare_csr into :hold_utc, :bare_lad;
            count_1 ++;
            if (bare_lad < bcomp_lad)
                  bcomp_lad = bare_lad;
            }
      count_1 --;
      bare_pop = bare_pop + (count_1 * hb_tab[hb_idx].hb_pop);
      if (count_1 >  0) hb_chk = 1;
      EXEC SQL CLOSE bare_csr;
      }

/************************************************************
*                CHECK FOR HARVEST BARE ADVANCE            *
************************************************************/

if (hb_chk > 0)
      {
      count_1 = 0; acomp_lad = 10; adv_lad = 10;
      if (hsb < 3)
            printf ("Harvest Bare Programmed\n");
       if (hsb > 1)
            fprintf (eout,"Harvest Bare Programmed\n");
      EXEC SQL DECLARE adv_chk_csr CURSOR FOR
            select
                  utc,
                  lad
            from air_tpfdd
            where ab_pod = :base_tab[base_idx].base_pod and
                  utc = :adv_utc[adv_idx];
      for (adv_idx = 0; adv_idx < adv_max; adv_idx ++)
            {
            EXEC SQL OPEN adv_chk_csr;
            while (sqlca.sqlcode == 0)
                  {
                  EXEC SQL FETCH adv_chk_csr into :hold_utc,
                        :adv_lad;
                  count_1 ++;
                  if (adv_lad < acomp_lad) acomp_lad = adv_lad;
                  }
            count_1 --;
            EXEC SQL CLOSE adv_chk_csr;
            }
      if (hsb > 1)
        if (count_1 == 0)
            fprintf (eout,"ERROR - Harvest Bare Advance Not"
```

```c
                              " Deployed\n\n");
        if (hsb < 3)
          if (count_1 == 0)
            printf ("ERROR - Harvest Bare Advance Not Deployed"
                          "\n\n");
        if (hsb > 1)
          if ((bcomp_lad <= acomp_lad) && (count_1 > 0))
            fprintf (eout,"ERROR - Harvest Bare Advance "
                          "Deployed Late\n\n");
          else fprintf (eout,"Harvest Bare Advance OK\n\n");
        if (hsb < 3)
          if ((bcomp_lad <= acomp_lad) && (count_1 > 0))
            printf ("ERROR - Harvest Bare Advance Deployed "
                          "Late\n\n");
          else printf ("Harvest Bare Advance OK\n\n");
      }
/************************************************************
*         Check For HE and HB Deployed to Same Base        *
************************************************************/

if (hsb > 1)
    if ((he_chk > 0) && (hb_chk > 0))
       fprintf (eout,"ERROR - HE and HB Deployed to Same "
                "Base\n\n");
if (hsb < 3)
    if ((he_chk > 0) && (hb_chk > 0))
       printf ("ERROR - HE and HB Deployed to Same Base"
                "\n\n");
if (hb_chk > 0) hold_tot = bare_pop;
hold_tot = hold_tot * 1.2;
if (he_chk > 0)
    printf ("Population Supported by HE is %d\n", hold_tot);
if ((hb_chk > 0) && (he_chk < 1))
    printf ("Population Supported by HB is %d\n", hold_tot);
if (hsb > 1)
    if (tot_pop > hold_tot)
       fprintf (eout,"ERROR - Insufficient Housekeeping "
                "(HE/HB)\n\n");
    else fprintf (eout,"Housekeeping OK\n\n");
if (hsb < 3)
    if (tot_pop > hold_tot)
       printf ("ERROR - Insufficient Housekeeping "
                "(HE/HB)\n\n");
    else printf ("Housekeeping OK\n\n");
}


/************************************************************
*         CHECK FOR MOSS IF HARVEST BARE WAS USED          *
************************************************************/

moss ()
{
```

78

```
if (hsb < 3)
printf ("------------MOSS CHECK------------\n\n");
if (hsb > 1)
fprintf(eout,"------------MOSS CHECK------------\n\n");
for (moidx = 0; moidx < momax; moidx ++)
    {
    count_1 = 0; hld_cnt = 0; hold_it = 0;
    EXEC SQL SELECT count(utc)
         into :hld_cnt
         from air_tpfdd
         where (ab_pod = :base_tab[base_idx].base_pod) and
                (utc = :motab[moidx].moa1 or
                 utc = :motab[moidx].moa2 or
                 utc = :motab[moidx].moa3);
    EXEC SQL SELECT count(utc)
         into :hold_it
         from air_tpfdd
         where ab_pod = :base_tab[base_idx].base_pod and
                (utc = :motab[moidx].mou1);
    count_1 = count_1 + (hold_it * motab[moidx].mou1n);
    EXEC SQL SELECT count(utc)
         into :hold_it
         from air_tpfdd
         where ab_pod = :base_tab[base_idx].base_pod and
                utc = :motab[moidx].mou2;
    count_1 = count_1 + (hold_it * motab[moidx].mou2n);
    EXEC SQL SELECT count(utc)
         into :hold_it
         from air_tpfdd
         where ab_pod = :base_tab[base_idx].base_pod and
                utc = :motab[moidx].mou3;
    count_1 = count_1 + (hold_it * motab[moidx].mou3n);
    EXEC SQL SELECT count(utc)
         into :hold_it
         from air_tpfdd
         where ab_pod = :base_tab[base_idx].base_pod and
                utc = :motab[moidx].mou4;
    count_1 = count_1 + (hold_it * motab[moidx].mou4n);
    if (hld_cnt > 0) form_ok = 1;
    if ((hsb > 1) && (hld_cnt > 0))
        if (count_1 < hld_cnt)
           fprintf (eout,"ERROR - Insufficient MOSS for"
                    " UTC(s) %s %s %s\n\n",motab[moidx].moa1
                    ,motab[moidx].moa2,motab[moidx].moa3);
        else fprintf (eout,"MOSS OK for UTC(s) %s %s %s\n\n",
                    motab[moidx].moa1,motab[moidx].moa2,
                    motab[moidx].moa3);
     if ((hsb < 3) && (hld_cnt > 0))
        if (count_1 < hld_cnt)
           printf ("ERROR - Insufficient MOSS for UTC(s)"
                    " %s %s %s\n\n",motab[moidx].moa1,
                    motab[moidx].moa2,motab[moidx].moa3);
```

```
            else printf ("MOSS OK for UTC(s) %s %s %s\n\n",
                    motab[moidx].moa1,motab[moidx].moa2,
                    motab[moidx].moa3);
        }
if (hsb > 1 && form_ok == 0)
        fprintf (eout,"MOSS Not Needed\n\n");
if (hsb < 3 && form_ok == 0)
        printf ("MOSS Not Needed\n\n");
}


/********************************************************************
*    CHECK FOR FUEL SUPPORT FOR EVERY FLYING SQUADRON          *
********************************************************************/

fuels ()
{

if (hsb < 3)
        printf ("------------ FUEL   CHECK   ------------\n\n");
if (hsb > 1)
        fprintf (eout,"------------FUEL   CHECK------------\n\n");
count_1 = 0; hld_cnt = 0; needs_fuel = 0;
for (fuel_idx = 0; fuel_idx < fuel_max; fuel_idx ++)
        {
        EXEC SQL SELECT count(utc)
                into :count_1
                from air_tpfdd
                where utc = :fuel_tab[fuel_idx].fuel_utc and
                        ab_pod = :base_tab[base_idx].base_pod;
        hld_cnt = hld_cnt + (count_1 *
                fuel_tab[fuel_idx].fuel_sqn);
        }
/*****        TOTAL UP NUMBER OF FLYING SQUADRONS    *****/
count_1 = 0;
EXEC SQL DECLARE fly_csr CURSOR FOR
        select
                air_tpfdd.utc
        from air_tpfdd, fly_sqn
        where air_tpfdd.ab_pod = :base_tab[base_idx].base_pod
                and air_tpfdd.utc = fly_sqn.fly_utc;
EXEC SQL OPEN fly_csr;
while (sqlca.sqlcode == 0)
        {
        EXEC SQL FETCH fly_csr into :hold_utc;
        count_1 ++;
        }
EXEC SQL CLOSE fly_csr;
count_1 --;
if (count_1 > 0) needs_fuel = 1;
if (hsb < 3)
        printf ("Flying Sqns = %d  Fuel Sqns = %d\n",
                count_1,hld_cnt);
```

```
if (hsb > 1)
    fprintf (eout,"Flying Sqns = %d  Fuel Sqns = %d\n",
              count_1,hld_cnt);
if (hsb > 1)          .
    if (hld_cnt < count_1)
       fprintf (eout,"ERROR - Insufficient Fuel Support"
              "\n\n");
    else fprintf (eout,"Fuel Support OK\n\n");
if (hsb < 3)
    if (hld_cnt < count_1)
      printf ("ERROR - Insufficient Fuel Support\n\n");
    else printf ("Fuel Support OK\n\n");
}


/************************************************************
*          MAKE SURE THAT EACH TFS HAS A STAMP            *
************************************************************/

chk_stamp ()
{

count_1 = 0;
hld_cnt = 0;
if (hsb < 3)
    printf ("------------- STAMP CHECK    -------------\n\n");
if (hsb > 1)
    fprintf (eout,"-------------STAMP CHECK------------\n\n");
EXEC SQL DECLARE sta_ac_csr CURSOR FOR
    select
           air_tpfdd.utc
    from air_tpfdd, stmp
    where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
          and (air_tpfdd.utc = stmp.sta_ac1 or
           air_tpfdd.utc = stmp.sta_ac2 or
           air_tpfdd.utc = stmp.sta_ac3 or
           air_tpfdd.utc = stmp.sta_ac4);
EXEC SQL OPEN sta_ac_csr;
while (sqlca.sqlcode == 0)
    {
    EXEC SQL FETCH sta_ac_csr into :hold_utc;
    count_1 ++;
    }
EXEC SQL CLOSE sta_ac_csr;
count_1 --;
EXEC SQL DECLARE sta_utc_csr CURSOR FOR
    select
           air_tpfdd.utc
    from air_tpfdd, stmp
    where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
          and (air_tpfdd.utc = stmp.stmp_utc);
EXEC SQL OPEN sta_utc_csr;
while (sqlca.sqlcode == 0)
```

```
            {
            EXEC SQL FETCH sta_utc_csr into :hold_utc;
            hld_cnt ++;
            }
      EXEC SQL CLOSE sta_utc_csr;
      hld_cnt --;
      if (count_1 == 0 && hld_cnt == 0)
            printf ("STAMP Not Needed\n\n");
      if ((hsb > 1) && (count_1 > 0))
            if (hld_cnt < count_1)
              fprintf (eout,"ERROR - Insufficient STAMP\n\n");
            else
                fprintf (eout,"STAMP OK \n\n");
      if ((hsb < 3) && (count_1 > 0))
            if (hld_cnt < count_1)
              printf ("ERROR - Insufficient STAMP\n\n");
            else
                printf ("STAMP OK\n\n");
      }


/******************************************************************
*          MAKE SURE THAT EACH TFS HAS A STRAPP                   *
******************************************************************/

chk_strapp ()
{

count_1 = 0;
hld_cnt = 0;
if (hsb < 3)
      printf ("------------STRAPP CHECK------------\n\n");
if (hsb > 1)
      fprintf (eout,"------------STRAPP CHECK------------"
                 "\n\n");
EXEC SQL DECLARE str_ac_csr CURSOR FOR
      select
            air_tpfdd.utc
      from air_tpfdd, strp
      where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = strp.str_ac1 or
              air_tpfdd.utc = strp.str_ac2 or
              air_tpfdd.utc = strp.str_ac3);
EXEC SQL OPEN str_ac_csr;
while (sqlca.sqlcode == 0)
      {
      EXEC SQL FETCH str_ac_csr into :hold_utc;
      count_1 ++;
      }
EXEC SQL CLOSE str_ac_csr;
count_1 --;
EXEC SQL DECLARE str_utc_csr CURSOR FOR
      select
            air_tpfdd.utc
```

```
            from air_tpfdd, strp
            where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
                  and (air_tpfdd.utc = strp.strp_utc);
EXEC SQL OPEN str_utc_csr;
while (sqlca.sqlcode == 0)
      {
      EXEC SQL FETCH str_utc_csr into :hold_utc;
      hld_cnt ++;
      }
EXEC SQL CLOSE str_utc_csr;
hld_cnt --;
if (count_1 == 0 && hld_cnt == 0)
      printf ("STRAPP Not Needed\n\n");
if ((hsb > 1) && (count_1 > 0))
      if (hld_cnt < count_1)
        fprintf (eout,"ERROR - Insufficient STRAPP\n\n");
      else
            fprintf (eout,"strapp OK\n\n");
if ((hsb < 3) && (count_1 > 0))
      if (hld_cnt < count_1)
        printf ("ERROR - Insufficient STRAPP\n\n");
      else
        printf ("STRAPP OK\n\n");
}


/********************************************************************
*   CHECK TIMING AND SEQUENCING OF UTCs WHERE ORDER IS          *
*   IMPORTANT - THE ORDER IS AS FOLLOWS 1. Site Survey          *
* 2. Aerial Port  3. Red Horse/Prime Beef  4. Housekeeping *
*   5. Combat Support Group (if HE)  6. Security               *
* THE ABOVE UNITS MUST ARRIVE IN THAT ORDER, AND BEFORE D-3*
********************************************************************/

sequence ()
{

if (hsb < 3)
      printf ("------------ CHECKING UTC SEQUENCING "
                  "------------\n\n");
if (hsb > 1)
      fprintf (eout,"------------CHECKING UTC SEQUENCING"
                  "------------\n\n");
EXEC SQL SELECT min(air_tpfdd.lad)
      into :survey_lad
      from air_tpfdd, survey
      where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = survey.survey_utc);
EXEC SQL SELECT min(air_tpfdd.lad)
      into :ap_lad
      from air_tpfdd, a_port
      where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = a_port.port_utc);
```

83

```
EXEC SQL SELECT min(air_tpfdd.lad)
      into :beef_lad
      from air_tpfdd, beef
      where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = beef.beef_utc);
if (he_chk > 0)
    {
    EXEC SQL SELECT min(air_tpfdd.lad)
          into :csg_lad
          from air_tpfdd, csg
          where (air_tpfdd.ab_pod =
                    :base_tab[base_idx].base_pod)
                and (air_tpfdd.utc = csg.csg_utc);
    EXEC SQL SELECT min(air_tpfdd.lad)
          into :hk_lad
          from air_tpfdd, eagle
          where (air_tpfdd.ab_pod =
                    :base_tab[base_idx].base_pod) and
                (air_tpfdd.utc = eagle.eagle_utc);
    }
if (hb_chk > 0)
    EXEC SQL SELECT min(air_tpfdd.lad)
          into :hk_lad
          from air_tpfdd, bare
          where (air_tpfdd.ab_pod =
                    :base_tab[base_idx].base_pod) and
                (air_tpfdd.utc = bare.bare_utc);
EXEC SQL SELECT min(air_tpfdd.lad)
      into :sp_lad
      from air_tpfdd, security
      where (air_tpfdd.ab_pod =
                :base_tab[base_idx].base_pod) and
            (air_tpfdd.utc = security.sp_utc);
if (hsb > 1)
    {
    fprintf (eout,"SITE SURVEY LAD IS     %d\n", survey_lad);
    fprintf (eout,"AERIAL PORT LAD IS     %d\n", ap_lad);
    fprintf (eout,"RED HORSE/BEEF LAD IS %d\n", beef_lad);
    fprintf (eout,"HOUSEKEEPING LAD IS    %d\n", hk_lad);
    if (he_chk > 0)
        fprintf (eout,"CSG LAD IS             %d\n", csg_lad);
    fprintf (eout,"SECURITY LAD IS        %d\n\n", sp_lad);
    if (survey_lad > ap_lad || survey_lad > -4)
        fprintf (eout,"ERROR - Site Survey Late, Out of"
                    "Sequence or Not Deployed\n\n");
      else
          fprintf (eout,"Site Survey Sequence OK\n\n");
    if (ap_lad > beef_lad || ap_lad > -4)
        fprintf (eout,"ERROR - Aerial Port Late or Out of"
                    " Sequence\n\n");
      else
          fprintf (eout,"Aerial Port Sequence OK\n\n");
    if (beef_lad > hk_lad || beef_lad > -4)
```

84

```c
            fprintf (eout,"ERROR - Prime Beef/Red Horse Late,"
                    " Out of Sequence, or Not Deployed\n\n");
        else
           fprintf (eout,"Prime Beef/Red Horse Sequence "
                    "OK\n\n");
      if (he_chk > 0)
         {
         if (hk_lad > csg_lad || hk_lad > -4)
            fprintf (eout,"ERROR - Housekeeping Late or "
                    "Out of Sequence\n\n");
            else
              fprintf (eout,"Housekeeping Sequence OK\n\n");
         if (csg_lad > sp_lad || csg_lad > -4)
            fprintf (eout,"ERROR - Combat Support Group "
                    "Late or Out of Sequence\n\n");
            else
              fprintf (eout,"Combat Support Group Sequence "
                    "OK\n\n");
         if ((sp_lad > -4) || (sp_lad < hk_lad))
            fprintf (eout,"ERROR - Security Late of Out "
                    "of Sequence "
                    "or Not Deployed\n\n");
            else
              fprintf (eout,"Security Sequence OK\n\n");
         }
      if (hb_chk > 0 && he_chk == 0)
         {
         if (hk_lad > sp_lad || hk_lad > -4)
            fprintf (eout,"ERROR - Housekeeping Late or Out"
                    " of Sequence\n\n");
            else
              fprintf (eout,"Housekeeping Sequence OK\n\n");
         if ((sp_lad > -4) || (sp_lad < hk_lad))
            fprintf (eout,"ERROR - Security Late or Out of"
                    " Sequence\n\n");
            else
                fprintf (eout,"Security Sequence OK\n\n");
         }
      }
   if (hsb < 3)
      {
      printf ("SITE SURVEY LAD IS     %d\n", survey_lad);
      printf ("AERIAL PORT LAD IS     %d\n", ap_lad);
      printf ("RED HORSE/BEEF LAD IS %d\n", beef_lad);
      printf ("HOUSEKEEPING LAD IS    %d\n", hk_lad);
      if (he_chk > 0)
          printf ("CSG LAD IS            %d\n", csg_lad);
      printf ("SECURITY LAD IS        %d\n\n", sp_lad);
      if (survey_lad > ap_lad || survey_lad > -4)
          printf ("ERROR - Site Survey Late, Out of Sequence"
                "or Not Deployed\n\n");
        else
          printf ("Site Survey Sequence OK\n\n");
```

```
        if (ap_lad > beef_lad || ap_lad > -4)
           printf ("ERROR - Aerial Port Late or Out of "
                    "Sequence\n\n");
          else
             printf ("Aerial Port Sequence OK\n\n");
        if (beef_lad > hk_lad || beef_lad > -4)
           printf ("ERROR - Prime Beef/Red Horse Late, Out of "
                    "Sequence, or Not Deployed\n\n");
          else
             printf ("Prime Beef/Red Horse Sequence OK\n\n");
        if (he_chk > 0)
           {
           if (hk_lad > csg_lad || hk_lad > -4)
              printf ("ERROR - Housekeeping Late or Out of "
                       "Sequence\n\n");
             else
                printf ("Housekeeping Sequence OK\n\n");
           if (csg_lad > sp_lad || csg_lad > -4)
              printf ("ERROR - Combat Support Group Late or "
                       "Out of Sequence\n\n");
             else
                printf ("Combat Support Group Sequence OK\n\n");
           if ((sp_lad > -4) || (sp_lad < hk_lad))
              printf ("ERROR - Security Late, Out of Sequence"
                       " or Not Deployed\n\n");
             else
                printf ("Security Sequence OK\n\n");
           }
        if (hb_chk > 0 && he_chk == 0)
           {
           if (hk_lad > sp_lad || hk_lad > -4)
              printf ("ERROR - Housekeeping Late or Out of "
                       "Sequence\n\n");
             else
                printf ("Housekeeping Sequence OK\n\n");
           if ((sp_lad > -4) || (sp_lad < hk_lad))
              printf ("ERROR - Security Late or Out of "
                       "Sequence\n\n");
             else
                 printf ("Security Sequence OK\n\n");
           }
        }
}

/*******************************************************************
*      CHECK TIMING OF AIRCRAFT SQUADRON ARRIVAL                   *
*            LAD MUST BE EARLIER THAN D-3                          *
********************************************************************/

acft_timing ()
{
if (hsb < 3)
    printf ("-----------CHECKING AIRCRAFT TIMING"
```

```c
                             "------------\n\n");
      if (hsb > 1)
          fprintf (eout,"------------   CHECKING AIRCRAFT TIMING"
                        "  ------------\n\n");
      EXEC SQL DECLARE acft_csr CURSOR FOR
          select
                air_tpfdd.utc,
                air_tpfdd.lad
          from air_tpfdd, fly_sqn
          where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
                and (air_tpfdd.utc = fly_sqn.fly_utc);
      EXEC SQL OPEN acft_csr;
      while (sqlca.sqlcode == 0)
            {
            EXEC SQL FETCH acft_csr into :hold_utc, :acomp_lad;
            if (sqlca.sqlcode < 1)
              {
              if (hsb > 1)
                if (acomp_lad > -4)
                  fprintf (eout,"ERROR - Late LAD for Aircraft UTC"
                               " %s\n",hold_utc);
                else
                  fprintf (eout,"LAD for Aircraft UTC %s OK\n",
                               hold_utc);
              if (hsb < 3)
                if (acomp_lad > -4)
                  printf ("ERROR - Late LAD for Aircraft UTC "
                               "%s\n",hold_utc);
                else
                  printf ("LAD for Aircraft UTC %s OK\n",hold_utc);
              }
            }
      if (hsb < 3)
          printf (" \n");
      if (hsb > 1)
          fprintf (eout," \n");
      EXEC SQL CLOSE acft_csr;
      }


/**********************************************************************
*      CHECK TIMING ON UTCs WHERE ORDER OF ARRIVAL IS               *
*      NOT IMPORTANT - Transient Maintenance, Communication,        *
*Fuels, Supply, Field Maintenance and Munitions Maintenence*
*             CHECK FOR ARRIVAL NO LATER THAN D-4                   *
**********************************************************************/

misc_timing ()
{

count_1 = 0;
if (hsb < 3)
    printf ("------------ CHECKING MISCELLANEOUS UTC TIMING "
               "------------\n\n");
```

87

```
if (hsb > 1)
    fprintf (eout,"----------CHECKING MISCELLANEOUS UTC "
                "TIMING----------\n\n");


/*********       CHECKING TRANS. MAINTENANCE TIMING    *******/
EXEC SQL SELECT min(air_tpfdd.lad)
    into :acomp_lad
    from air_tpfdd, t_maint
    where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = t_maint.tm_utc);
if (hsb > 1)
    {
    fprintf (eout,"Transient Maintenance LAD is %d\n",
                acomp_lad);
    if (acomp_lad > -4)
        fprintf (eout,"ERROR - Transient Maintenance "
                    "Deployed Late\n\n");
      else fprintf (eout,"Transient Maintenance Timing "
                    "OK\n\n");


    }
if (hsb < 3)
    {
    printf ("Transient Maintenance LAD is %d\n",acomp_lad);
    if (acomp_lad > -4)
        printf ("ERROR - Transient Maintenance Deployed "
                    "Late\n\n");
      else printf ("Transient Maintenance Timing OK\n\n");
    }

/********       CHECKING COMMUNICATIONS TIMING    ********/
EXEC SQL SELECT min(air_tpfdd.lad)
    into :acomp_lad
    from air_tpfdd, comm
    where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
            and (air_tpfdd.utc = comm.comm_utc);
if (hsb > 1)
    {
    fprintf (eout,"Communication LAD is %d\n",acomp_lad);
    if (acomp_lad > -4)
        fprintf (eout,"ERROR - Communication Deployed "
                    "Late\n\n");
      else fprintf (eout,"Communication Timing OK\n\n");
    }
if (hsb < 3)
    {
    printf ("Communication LAD is %d\n",acomp_lad);
    if (acomp_lad > -4)
        printf ("ERROR - Communication Deployed Late\n\n");
      else printf ("Communication Timing OK\n\n");
    }
```

```
/************    CHECKING FUELS TIMING    ************/
if (needs_fuel > 0)
     {
     EXEC SQL SELECT min(air_tpfdd.lad)
          into :acomp_lad
          from air_tpfdd, fuels
          where (air_tpfdd.ab_pod =
               :base_tab[base_idx].base_pod) and
               (air_tpfdd.utc = fuels.fuels_utc);
     if (hsb > 1)
          {
          fprintf (eout,"Fuel Support LAD is %d\n",acomp_lad);
          if (acomp_lad > -4)
             fprintf (eout,"ERROR - Fuel Support Deployed "
                         "Late\n\n");
           else fprintf (eout,"Fuel Support Timing OK\n\n");
          }
     if (hsb < 3)
          {
          printf ("Fuel Support LAD is %d\n",acomp_lad);

          if (acomp_lad > -4)
             printf ("ERROR - Fuel Support Deployed "
                         "Late\n\n");
           else printf ("Fuel Support Timing OK\n\n");
          }
     }

/***********    CHECKING SUPPLY TIMING    ***********/

EXEC SQL SELECT min(air_tpfdd.lad)
     into :acomp_lad
     from air_tpfdd, supply
     where (air_tpfdd.ab_pod = :base_tab[base_idx].base_pod)
          and (air_tpfdd.utc = supply.supply_utc);
if (hsb > 1)
     {
     fprintf (eout,"Supply Support LAD is %d\n",acomp_lad);
     if (acomp_lad > -4)
        fprintf (eout,"ERROR - Supply Support Deployed "
                   "Late\n\n");
      else fprintf (eout,"Supply Support Timing OK\n\n");
      }
if (hsb < 3)
     {
     printf ("Supply Support LAD is %d\n",acomp_lad);
     if (acomp_lad > -4)
        printf ("ERROR - Supply Support Deployed Late\n\n");
      else printf ("Supply Support Timing OK\n\n");
      }
```

```
/*****    CHECKING FIELD AND MUNITIONS MAINTENANCE    *****/
for (fldidx = 0; fldidx < fldmax; fldidx ++)
    {
    count_1 = 0;
    EXEC SQL SELECT count(utc)
        into :count_1
        from air_tpfdd
        where (ab_pod = :base_tab[base_idx].base_pod) and
            (utc = :fldtab[fldidx].fldac1 or
             utc = :fldtab[fldidx].fldac2 or
             utc = :fldtab[fldidx].fldac3);
    if (count_1 > 0)
        {
        EXEC SQL SELECT min(lad)
            into :acomp_lad
            from air_tpfdd
            where (ab_pod = :base_tab[base_idx].base_pod) and
                (utc = :fldtab[fldidx].fldutc1 or
                 utc = :fldtab[fldidx].fldutc2 or
                 utc = :fldtab[fldidx].fldutc3);
        if (hsb > 1)
            {
            fprintf (eout,"Field Maintenance LAD for %s "
                    "is %d\n",fldtab[fldidx].fldtype,
                    acomp_lad);
            if (acomp_lad > -4)
                fprintf (eout,"ERROR - Field Maintenance "
                        "Late or Not Deployed\n\n");
             else fprintf (eout,"Field Maintenance OK\n\n");
             }
        if (hsb < 3)
            {
            printf ("Field Maintenance LAD for %s is %d\n",
                    fldtab[fldidx].fldtype, acomp_lad);
            if (acomp_lad > -4)
                printf ("ERROR - Field Maintenance Late or "
                        "Not Deployed\n\n");
             else printf ("Field Maintenance OK\n\n");
             }
        }
    }
for (munidx = 0; munidx < munmax; munidx ++)
    {
    count_1 = 0;
    EXEC SQL SELECT count(utc)
        into :count_1
        from air_tpfdd
        where (ab_pod = :base_tab[base_idx].base_pod) and
            (utc = :muntab[munidx].munac1 or
             utc = :muntab[munidx].munac2 or
             utc = :muntab[munidx].munac3);
    if (count_1 > 0)
        {
```

90

```
       EXEC SQL SELECT min(lad)
           into :acomp_lad
           from air_tpfdd
           where (ab_pod = :base_tab[base_idx].base_pod) and
                 (utc = :muntab[munidx].munutc1 or
                  utc = :muntab[munidx].munutc2 or
                  utc = :muntab[munidx].munutc3);
   if (hsb > 1)
       {
       fprintf (eout,"Munitions Maintenance LAD for "
                         "%s is %d\n",muntab[munidx].muntype,
                         acomp_lad);
       if (acomp_lad > -4)
           fprintf (eout,"ERROR - Munitions Maintenance"
                         " Late or Not Deployed\n\n");
        else fprintf (eout,"Munitions Maintenance "
                            "OK\n\n");
       }
   if (hsb < 3)
       {
       printf ("Munitions Maintenance LAD for %s is "
                 %d\n",muntab[munidx].muntype, acomp_lad);
       if (acomp_lad > -4)
           printf ("ERROR - Munitions Maintenance Late "
                        "or Not Deployed\n\n");
        else printf ("Munitions Maintenance OK\n\n");
        }
    }
  }
 }
```

Appendix C: Data File Maintenance

Overview

There were a number of new data files that were added to the JPLAN database as a result of the JPLAN evaluation system. There is not however, a special program for the user to maintain these files, although a recommendation of this study is that one be developed as soon as possible. Until such a program can be developed, the user will have to use the Query-By-Forms (QBF) module of the interactive PC INGRES system to add, delete, modify, or review records in the evaluation system data files.

This appendix will provide the user with an introduction to the QBF portion of interactive INGRES. It is not intended to be a full tutorial for that system. For a complete QBF tutorial, the user should refer to chapters 6 through 11 of the PC INGRES Reference Guide (12:6.1-11.5). This appendix will explain to the user how to access interactive INGRES, how to choose a file for updating, how to access the QBF module of INGRES, and how to select the proper update function.

In addition to providing an introduction to INGRES QBF, this appendix will also provide information about each of the evaluation system data files, such as file names, field names in each file, what each field contains, whether the field is alphanumeric or integer, and record layout of each

file. An alphanumeric field is one which can contain either letters or integer numbers, while an integer field contains only whole numbers, either positive or negative.

Interactive INGRES and QBF

In explaining how to get into interactive INGRES and how to use the QBF portion of INGRES, commands that the user must enter will be identified in capital letters. When [RETURN] is seen, it means the user should hit the return key. All other commands are designated by parenthesis. The following is a step-by-step overview on how to use the QBF sub-system of PC INGRES:

1.   From the DOS C: prompt, make INGRES memory resident by entering (ADDINGRES) [RETURN].

2.   At the C: prompt again, begin interactive INGRES by entering (INGRES JPLAN) [RETURN].

3.   An introductory screen will appear with a menu of options across the top of the screen. The 'tables' option will be highlighted. The user should enter [RETURN].

4.   Another screen will appear with a menu of options across the top, and a list of file names in the middle of the screen. The cursor will be over the first file name. Using the up and/or down arrow keys, highlight the file name that needs to be updated. Press the (F2) key to move the cursor to the menu options on the top of the screen. Whenever the cursor is in the middle of the screen, as it

was at the beginning of this screen, pressing (F2) will get

the user to the menu bar at the top of the screen.

5.   Enter (Q) to enter the QBF sub-system.

6.   The system will then ask if the user wants

Simple_Fields or Table_Fields.   Enter (T) for Table_Fields.

This will allow the user to work with and/or view more than

one record at a time.   Simple_Fields allows the user to work

on and view only one record at a time.

7.   The QBF menu will appear across the top of the screen.

To add a new record to the selected file, the user should

enter A for append.   To change or delete existing records in

the selected file, the user should select (U) for update.

To just view data, the user should choose (R) for retrieve.

To select a new file to work with in QBF, enter (N).   Each

of these functions will be addressed seperately.

8.   APPEND.

      a.   Enter (A) to add a new record, or several new

      records to the file.

      b.   A screen will appear with the field names for the

      records in the chosen file.   Enter the information for

      the record(s) to be added.   To move the cursor between

      columns and lines, use the arrow or tab keys.

      c.   To save the records that have been added, press

      either (F9) alone, or press (F2) and then press (G).

      This will append the records to the end of the file.

      d.   Press (F10) to return to the QBF menu when

      additions are complete

9. UPDATE.

    a. Enter (U) to modify or delete an existing record.

    b. The same column and field name screen as described in 8b above will appear. Enter some identifying characteristic of the record you want to change, such as the UTC, in the appropriate column. Wildcards such as (*) are acceptable. For example, entering a (U*) in the UTC field will cause the system to retrieve all UTCs in the file that begin with U. If (UF*) were entered, it would find all UTCs that begin with UF. After the identifying characteristic is entered, press (F9), or press (F2) and (G).

    c. All records meeting the identifying criteria will be posted on the screen. Using the arrow and tab keys to position the cursor, make any necessary changes. After changes are complete, press (F2) and (S) to save the changes.

    d. To delete a record, position the cursor on the record to be deleted, and press (F2) and (D). A new menu will appear at the top of the screen. Press (R).

    e. After all changes are complete, press (F10) to return to the QBF menu.

10. RETRIEVE.

    a. To review data, enter (R) from the QBF menu.

b. The same screen seen in the other functions will appear. Enter selection criteria and press either (F9) or (F2) and (G).

c. To return to the QBF screen, press (F10).

11. NEWFILE.

a. To work on a different file in the QBF function, enter (N).

b. The system will ask for the name of the next file to work on. Enter the filename and [RETURN].

12. To exit from INGRES, and get back to DOS from the QBF menu, press (F10) twice.

Evaluation System Database Overview

The previous section of this appendix was an overview on the mechanics of maintaining the evaluation system data files. Before the user attempts to make any changes to the evaluation system data files, he should have an understanding of what is in each file.

Before that however, the user should have a clear understanding of some identifying terms, such as what is meant by a data file, a record and a data field.

A data file is a collection of related information which is made up of individual lines of information, which are in turn made up of individual bits of information. The phone book, which is an example of a data file, is made up of individual lines of information which include names, addresses and phone numbers. Each of these lines is called

a rec..rd, while the individual bits of information that represent the names, addresses and phone numbers are called data fields or fields. Therefore, a file (the phone book) is made up of records (all the information on one person included in the phone book), which are, in turn, made up of individual data fields (name, address, phone number).

Evaluation System Data File Descriptions

The following paragraphs describe each data file and the data fields in all of the data files created for the evaluation system. It will give the name of each field, what each field is used for, what information should go into that field, and what type of information (integer, or alphanumeric) is in that file. All alphanumeric fields are five character positions.

A_PORT. The A_PORT file identifies aerial port UTCs to the evaluation system, and the daily cargo handling capacity of each. Each record is made up of two fields. One is called PORT_UTC, which is alphanumeric, and should contain aerial port UTCs. The other is called PORT_TONS. This is an integer field which should contain the daily cargo handling capacity of its accompanying UTC. This file should contain a record for every aerial port UTC that the students can program in the exercise. The records on the file can be in any order. Record layout is as follows:

PORT_UTC   Alphanumeric

PORT_TONS   Integer

STMP. This file identifies to the system every STAMP UTC, and what aircraft UTCs need that particular STAMP. Each record is made up of five alphanumeric fields. The first is called STMP_UTC, and is the UTC of the STAMP. The other four are called STA_AC#, where # is a number from 1 to 4. These are the aircraft UTCs that need that particular STAMP. There should be a record in the file for every STAMP UTC, but there does not necessarily have to be an aircraft UTC to fill in all four STA_AC# fields. For example, STAMP UTC HHJSC is only needed by three aircraft UTCs, so in the file for that record, STA_AC4 is left blank. Order is not important. The record layout is as follows:

STMP_UTC	Alphanumeric
STA_AC1	Alphanumeric
STA_AC2	Alphanumeric
STA_AC3	Alphanumeric
STA_AC4	Alphanumeric

STRP. This file identifies STRAPP UTCs to the system, and the aircraft squadrons that need them. It is identical to the STMP file, except that the field names are STR_UTC and STR_AC#, where # is a number from 1 to 3. The record layout is as follows:

STRP_UTC	Alphanumeric
STRP_AC1	Alphanumeric
STRP_AC2	Alphanumeric
STRP_AC3	Alphanumeric

EAGLE. This file identifies Harvest Eagle UTCs to the
evaluation system, and the base population each will
support. Each record contains two fields. One is an
alphanumeric field called EAGLE_UTC, which should contain
the Harvest Eagle UTC. The other field is an integer field
called EAGLE_POP, and should contain the base population its
accompanying UTC can support. There should be one record on
file for every Harvest Eagle. Record order is not
important. Record layout is as follows:

>   EAGLE_UTC   Alphanumeric

>   EAGLE_POP   Integer

BARE. This file identifies Harvest Bare UTCs and the
base populations they can support to the evaluation system.
Each record contains two fields, one alphanumeric, and one
integer. The alphanumeric field is called BARE_UTC and
should contain a Harvest Bare UTC. The integer field is
called BARE_SPT and should contain the population its
accompanying Harvest Bare UTC can support. There should be
one record on file for every Harvest Bare UTC. Order is not
important. Record layout is as follows:

>   BARE_UTC   Alphanumeric

>   BARE_SPT   Integer

SUPPLY. This file identifies Supply Support UTCs and
the base populations they can support to the evaluation
system. Each record is made up of two fields. One is
alphanumeric, and is called SUPPLY_UTC. It contains the UTC
of a supply support unit. The other is integer, and

contains the base population its accompanying UTC can support. There should be a record on file for every supply support unit UTC. Order is not important. Record layout is as follows:

SUPPLY_UTC   Alphanumeric

SUPPLY_SPT   Integer

FUELS. This file identifies POL support UTCs and the number of flying squadrons each can support to the evaluation system. Each record contains two fields, an alphanumeric and an integer field. The alphanumeric field is called FUELS_UTC, and contains the UTC of the POL support unit. The integer field is called FUELS_SQN and contains the number of flying squadrons its accompanying UTC can support. There should be one record on file for every POL support UTC. Order of records is not important. Record layout is as follows:

FUELS_UTC   Alphanumeric

FUELS_SQN   Integer

CSG. This file identifies to the evaluation system Combat Support Group UTCs and the base population each can support. There are two fields in each record. First is an alphanumeric field called CSG_UTC, contains the UTC of the Combat Support Group. The other is an integer field called CSG_SPT, which is the population its accompanying CSG UTC can support. There should be a record on file for every

Combat Support Group UTC.   Record order is not important.
Record layout is as follows:

> CSG_UTC   Alphanumeric
>
> CSG_SPT   Integer

T MAINT.   This file identifies Transient Maintenance
support UTCs, and the base population they can support to
the evaluation system.   Each record contains two fields.
There is an alphanumeric field called TM_UTC which contains
the UTC of each Transient Maintenance package.   The other
field is an integer field called TM_SPT which contains the
base population its accompanying UTC can support.   There
must be a record on file for every Transient Maintenance
UTC.   Record order within the file is not important.   Record
layout is as follows:

> TM_UTC   Alphanumeric
>
> TM_SPT   Integer

MATH.   This file identifies MATH UTCs to the evaluation
system.   It contains one alphanumeric field called MATH_UTC
which contains the UTC of a MATH.   There should be one
record on file for every MATH UTC.   Order is not important.
Record layout is as follows:

> MATH_UTC   Alphanumeric

ARMY.   This file contains one record which is made up
of one integer field.   The field is called GRUNTS, and it
contains the preload Army population for the entire JPLAN

101

database. There should be only one record in this file.
Record layout is as follows:

GRUNTS   Integer

ADV. This file identifies the Harvest Bare Advance
UTCs to the evaluation system. Each record contains one
alphanumeric field called HB_ADV_UTC which contains the UTC
of a Harvest Bare Advance unit. There should be one record
for every Harvest Bare Advance UTC. Order is not important.
Record layout is as follows:

HB_ADV_UTC   Alphanumeric

HB MOSS. This file identifies to the evaluation system
all Flying Squadrons that require MOSS support, all MOSS
UTCs that can support the accompanying Flying Squadron UTCs,
and the number of Flying Squadrons each MOSS UTC can
support. Each record contains 11 fields, 7 alphanumeric and
4 integer. First are three alphanumeric fields called
MOAC#, where # is a number between one and three. These
fields identify up to three Flying Squadron UTCs that
require MOSS support. The following eight fields are in
four pairs of one alphanumeric, one integer each. the
alphanumeric fields are called MOUTC#, where # is a number
between one and four. These UTCs represent MOSS UTCs that
can support any of the three aircraft UTCs. The integer
fields are called MONO#, where # is also a number between
one and four. These fields represent the number of flying
squadrons its accompanying MOSS UTC can support.

Each record says that the up to three different aircraft UTCs can be supported by the up to four MOSS UTCs. There does not have to be three aircraft UTCs identified on each record, but there must be at least one. There also does not have to be four MOSS UTCs entered in each record, but there must be at least one with the accompanying number of flying squadrons it can support. An example of an actual HB_MOSS record may clarify this. One record has UTCs 3FJDC and 3FJDD identified as flying squadron UTCs. It also has MOSS UTCs XFFYE, XFFYF, XFFYG, XFFYH identified as MOSS units. This mean that either of the two identified flying UTCs can have their MOSS support requirements satisfied by any one of the four MOSS UTCs. All flying squadron UTCs that require MOSS support and all MOSS UTCs should be in at least one record. Record order is not important. Record layout is as follows:

MOAC1	Alphanumeric
MOAC2	Alphanumeric
MOAC3	Alphanumeric
MOUTC1	Alphanumeric
MONO1	Integer
MOUTC2	Alphanumeric
MONO2	Integer
MOUTC3	Alphanumeric
MONO3	Integer
MOUTC4	Alphanumeric
MONO4	Integer

FLY SQN. This file identifies all Flying Squadron UTCs to the evaluation system. Each record has one alphanumeric field called FLY_UTC which contains the UTC of a flying squadron. There should be a record on file for every Flying Squadron UTC. Record order is not important. Record layout is as follows:

> FLY_UTC   Alphanumeric

SURVEY. This file identifies all Site Survey UTCs to the evaluation system. There should be one record on file for every Site Survey UTC. Record order is not important. Record layout is as follows:

> SURVEY_UTC   Alphanumeric

BEEF: This file contains all Prime Beef and Red Horse UTCs. There should be one record on file for every Prime Beef and Red Horse UTC. Record Order is not important. Record layout is as follows:

> BEEF_UTC   Alphanumeric

SECURITY: This file identifies Security UTCs to the evaluation system. There should be one record on file for every Security UTC. Record order is not important. Record layout is as follows:

> SP_UTC   Alphanumeric

COMM: This file identifies all Communications UTCs to the evaluation system. There should be one record on file

for every Communications UTC. Record order is not important, and the record layout is as follows:

> COMM_UTC Alphanumeric

FMAINT: This file identifies to the evaluation system all aircraft UTCs that need Field Maintenance support, and the UTCs of the appropriate Field Maintenance units. Each record can contain up to three aircraft UTCs, up to three Field Maintenance UTCs, and a clear text aircraft identifier. Each aircraft UTC in a record should be able to be supported by the Field Maintenance UTCs identified in the same record. Every aircraft UTC that needs Field Maintenance support, and the Field Maintenance UTCs that can support them should be in the file. Record order is not important. The record layout is as follows:

> FTYPE Alphanumeric
>
> FAC_UTC1 Alphanumeric
>
> FAC_UTC2 Alphamuneric
>
> FAC_UTC3 Alphanumeric
>
> FMUTC1 Alphanumeric
>
> FMUTC2 Alphanumeric
>
> FMUTC3 Alphanumeric

MMAINT: This file is identical to the FMAINT file except it identifies aircraft UTCs that need Munitions Maintenance support, and the UTCs of Munitions Maintenance units. Record layout is as follows:

> MTYPE Alphanumeric
>
> MAC_UTC1 Alphanumeric

105

MAC_UTC2	Alphamuneric
MAC_UTC3	Alphanumeric
MMUTC1	Alphanumeric
MMUTC2	Alphanumeric
MMUTC3	Alphanumeric

Bibliography

1. Davis, Michael W. <u>Applied Decision Support</u>. Englewood Cliffs NJ: Prentice Hall, 1988.

2. Department of Defense. <u>Armed Forces Staff College Pub 1: Joint Staff Officers Guide</u>. National Defense University, Armed Forces Staff College, Norfolk VA, 1986.

3. Department of the Air Force. <u>Air Command and Staff College Basic Deployment Data Handbook</u>. Air University, Maxwell AFB AL.

4. Department of the Air Force. <u>Joint Operation Planning Exercise - (JPLAN)</u>. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

5. Department of the Air Force. <u>Syllabus of Instruction - LOG 299 Combat Logistics</u>. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1987.

6. Department of the Air Force. <u>War Planning: USAF Mobility Planning</u>. AFR 28-4. Washington: HQ USAF, 16 November 1978.

7. Dragich, Lt Col Dennis P., Assistant Professor of Logistics Management. Personal interviews. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 through February 1988.

8. Jansen, Capt James R. <u>JPLAN Monitor Users Guide</u>. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1987.

9. -----. <u>Redesign of the Joint Planning Exercise (JPLAN)</u>. MS Thesis, AFIT/GCS/ENG/87D-15. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1987 (AD-A189599).

10. Jansen, Capt James R., AFIT Student. Telephone interview. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 17 December 1987.

11. Melsha, Capt Joel E., Instructor of Logistics Management. Personal interviews. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February through July 1988.

12. PC Ingres Reference Guide. Relational Technology, Inc., May 1987.

13. Roth, Capt Mark, Assistant Professor of Computer Sciences. Personal interviews. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February through July 1988.

14. Schildt, Herbert. Using Turbo C. Berkeley, CA: Borland-Osborne/McGraw-Hill, 1988.

Capt Chester J. Jean, Jr. was born on ████████████
██

████████ where he graduated from Metuchen High School in
1969. He attended James Madison University in Harrisonburg,
Virginia, graduating in 1974 with a Bachelor of Science
degree in Health and Physical Education. He enlisted in the
U.S. Air Force in 1976 and served as a computer operator for
the 727 Tactical Control Squadron at Bergstrom AFB, Texas.
He received his commission from the Air Force Officer
Training School in 1978. Following computer programmer
training at Keesler AFB, Mississippi, he was assigned to the
4501 Computer Services Squadron, Headquarters, Tactical Air
Command, Langley AFB, Virginia as a Contingency Support
Analyst, and as Chief of the Command and Control Systems
Branch. Capt Jean was then assigned to Headquarters,
Pacific Air Forces, Hickam AFB, Hawaii as a Logistics Plans
Staff Officer, and as Chief of Logistics Requirements
Systems. He entered the School of Systems and Logistics,
Air Force Institute of Technology in 1987.

██

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GLM/LSM/88S-39	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION School of Systems and Logistics	6b. OFFICE SYMBOL (If applicable) AFIT/LSM	7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Automated Evaluation System For The Joint Planning (JPLAN) Exercise System

12. PERSONAL AUTHOR(S)
Chester J. Jean, Jr., B.S., Capt, USAF

13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1988 September	15. PAGE COUNT 120

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Applications, Analyzers, Military Training, Computer Simulation
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: AUTOMATED EVALUATION SYSTEM FOR THE JOINT PLANNING (JPLAN) EXERCISE SYSTEM

Thesis Advisor: Joel E. Melsha, Captain, USAF
Instructor of Logistics Management

Approved for public release IAW AFR 190-1.

WILLIAM A. MAUER   17 Oct 88
Associate Dean
School of Systems and Logistics
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Joel E. Melsha, Captain, USAF	22b. TELEPHONE (Include Area Code) (513) 255-5023	22c. OFFICE SYMBOL AFIT/LSM

DD Form 1473, JUN 86	Previous editions are obsolete.	SECURITY CLASSIFICATION OF THIS PAGE

Abstract

The purpose of this thesis was to develop a computer program which would apply the same evaluation criteria to student performance in the Joint Planning (JPLAN) exercise simulation system that the instructor previously had to apply manually.

JPLAN, which is used in Combat Logistics (LOG 299), an Air Force Institute of Technology (AFIT) Professional Continuing Education (PCE) course, is a computerized simulation of the real-world Joint Operation Planning System (JOPS). JOPS is the Department of Defense system used for conducting joint planning.

In JPLAN, students are presented with a partially completed Operations Plan based on a fictitious scenario in which planning for the defense of Iguana, an American ally, is underway. Students must complete the plan by programming Combat Support Forces and Combat Services Support Forces to support pre-programmed combat forces.

Following completion of the exercise, student performance is evaluated by the instructor for shortfalls and discrepancies. Factors evaluated are; sufficient housekeeping, supply, combat support group, and transient maintenance to support each base's population; sufficient STAMP, STRAPP, fuel, field maintenance, and munitions maintenance for all flying squadrons; sufficient aerial port support to handle the average daily tonnage coming into each base; two hospitals in theater; proper sequencing of the UTCs that must arrive in a given sequence; and arrival timing for the UTCs that must arrive prior to the start of flying operations on Day D-3.

A computer program was developed to take the JPLAN data files that comprise the students output, and evaluate them, applying the criteria described above. With this program, evaluation was reduced from over 3 hours using the manual method, to approximately 30 minutes.

The program was written using Microsoft C as the programming language and INGRES as the database management system, the same developmental software used in the JPLAN exercise system.